2026 59th IEEE/ACM International Symposium on Microarchitecture (MICRO)



SCARF: A Scene-Adaptive Depth-Guided G-3DGS Encoder Accelerator with Semantic Reuse and Fused Dataflow

Zirui Ma[1,2], Zhihua Fan[1,2,*], Haibin Wu[1,2], Wenxing Li[1,2], Fulin Zhang[3,1],
Dongrui Fan[1], Xiaochun Ye[1,2], and Wenming Li[1,2,*]
[1]Institute of Computing Technology, Chinese Academy of Sciences, Beijing, China
[2]University of Chinese Academy of Sciences, Beijing, China
[3]School of Advanced Interdisciplinary Sciences, University of Chinese Academy of Sciences, Beijing, China
{mazirui24s, fanzhihua, wuhaibin, liwenxing}@ict.ac.cn
{fandr, yexiaochun, liwenming}@ict.ac.cn; zhangfulin20@mails.ucas.ac.cn
[*]Corresponding authors: Zhihua Fan and Wenming Li.

***Abstract*—Depth-guided, cost-volume-based Generalizable 3D Gaussian Splatting (G-3DGS) encoders reconstruct scenes as sets of Gaussian primitives directly from multi-view images without requiring per-scene optimization, thereby facilitating feed-forward novel-view synthesis on edge computing platforms. However, their shared encoder architecture presents two primary bottlenecks. First, the per-pixel depth search disregards semantic similarity within the 2D feature space, causing depth search operations to account for over 65% of memory traffic due to redundant exhaustive candidate scans. Second, the sequential pipeline of depth prediction followed by Gaussian generation neglects local 3D geometric continuity, resulting in more than 52% of generated Gaussians being redundant prior to instantiation. In this paper, we introduce SCARF, a dataflow-co-designed accelerator that aligns each bottleneck with a corresponding scene-level regularity. To mitigate the first bottleneck, Feature Similarity Depth Reuse reconceptualizes depth search as a problem of semantic locality: depth consistency is determined by affinity in feature space rather than physical address proximity. It transforms exhaustive per-pixel depth searches into anchor-centered local searches. To overcome the second bottleneck, Scene-Adaptive Early Sparsification disrupts the conventional serial depth-to-materialization boundary by probing each tile to infer local scene regularity. It then selectively activates only the minimally sufficient generation pathway, eliminating redundant Gaussian computations before primitive instantiation. On a 27.8 square millimeter 28 nm post-layout design, SCARF delivers a 2.94-fold geometric-mean end-to-end speedup with maximum PSNR degradation below 0.072 percent, 2.23 inferences per joule, and 34.5-fold GPU energy efficiency.**

***Index Terms*—Generalizable 3D Gaussian Splatting, Domain-Specific Accelerator, Dataflow Optimization**

I. Introduction

Latency-sensitive edge applications, including robotic navigation [1], augmented reality [2], and autonomous driving [3], require novel-view synthesis that generates scene representations directly from captured images without per-scene optimization [4]–[6]. 3D Gaussian Splatting (3DGS) [7]–[9] enables real-time rendering through tile-based differentiable rasterization. However, traditional 3DGS is scene-specific and uses Structure-from-Motion initialization [10] followed by thousands of gradient iterations to construct the Gaussian set. Consequently, reconstruction takes hours per scene, making traditional 3DGS impractical for feed-forward edge deployment. As illustrated in Fig. 1, depth-guided, cost-volume-based Generalizable 3D Gaussian Splatting **(G-3DGS)** encoders [11]–[13] address this limitation by generating Gaussian primitives from multi-view images in one forward pass through cross-view matching, cost-volume depth estimation, and per-pixel regression. This preserves cross-scene generalization without retraining while replacing scene-specific iteration, reducing latency by three to four orders of magnitude relative to scene-specific optimization [14].

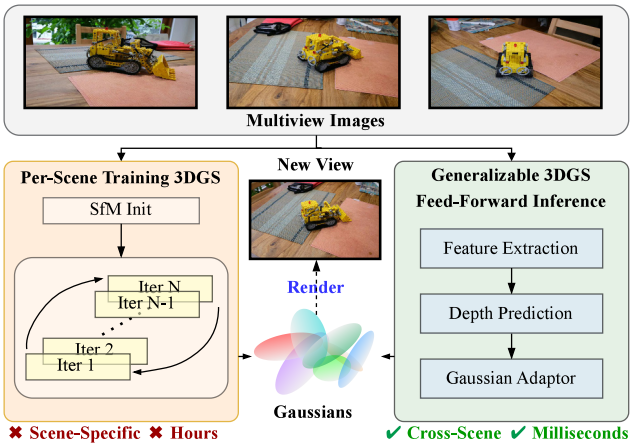


Fig. 1. G-3DGS vs. per-scene training 3DGS.

Despite software progress, these encoders share a common cost-volume dataflow that reveals two critical bottlenecks be-

This work was supported by the National Key R&D Program of China (Grant No. 2023YFB4503500), the National Natural Science Foundation of China (Grant No. 62502498), the Beijing Natural Science Foundation (Grant No. L234078), and the Innovation Funding of ICT, CAS (Grant No. E661120).

2379-3155/26/$31.00 ©2026 IEEE
DOI 10.1109/MICRO67795.2026.00172

yond the reach of software optimization. First, depth search is conducted independently for each pixel, disregarding semantic similarities among pixels. As a result, pixels with similar appearances, and thus nearly identical depth values, still undergo full candidate evaluation. This approach causes the depth search to dominate memory bandwidth usage, accounting for over 65% of total memory traffic. Second, the sequential pipeline comprising depth prediction and Gaussian generation fails to exploit the continuity inherent in local 3D geometry. Under stringent quality requirements, over 52% of generated Gaussians are redundant, yet the dataflow cannot eliminate them before generation, causing avoidable computation.

The two bottlenecks stem from two underexplored scene-level regularities: semantically similar pixels tend to share consistent depth estimates, and geometrically continuous surfaces produce redundant Gaussian representations. In this paper, we propose **SCARF**, a dataflow-co-designed accelerator that exploits semantic similarity during depth search and local geometric continuity during Gaussian materialization to reduce redundant memory accesses and Gaussian computation. SCARF combines **Feature Similarity Depth Reuse (FSDR)** and **Scene-Adaptive Early Sparsification (SAES)**. FSDR contracts full depth-candidate scans into semantic-guided local searches, while SAES selects the minimally sufficient generation path before Gaussian materialization. Our principal contributions are as follows:

(i) We introduce **Feature Similarity Depth Reuse**, which reconceptualizes per-pixel depth search as semantic locality rather than physical proximity. FSDR uses semantic signatures from locality-sensitive hashing (LSH) to index content-addressable memory (CAM). On a cache hit, it replaces the full candidate scan with a localized neighborhood search, reducing cross-view feature-sampling traffic.

(ii) We design **Scene-Adaptive Early Sparsification**, which disrupts the traditional serial boundary between depth estimation and Gaussian materialization. With a small probe set, SAES evaluates per-tile feature and depth consistency and activates only the minimally sufficient generation path, pruning redundant Gaussian computations while the resulting impact on quality remains controllable.

(iii) We develop **SCARF**[1], a $27.8\,\mathrm{mm}^2$ post-layout accelerator in 28 nm that integrates FSDR and SAES for depth-guided, cost-volume-based G-3DGS encoder inference. Evaluation shows a $2.94\times$ geometric mean speedup with maximum peak signal-to-noise ratio (PSNR) degradation below 0.072% and a $34.5\times$ energy-efficiency improvement over the Jetson Orin NX edge GPU.

II. Background and Motivation

A. Generalizable 3DGS Encoders

Generalizable 3D Gaussian Splatting (G-3DGS) encoders [11]–[13], [15]–[17] generate scene representations as sets of Gaussian primitives directly from multi-view images in a single feed-forward pass. Existing G-3DGS encoders fall into two categories. Geometry-consistent depth-guided encoders first estimate per-pixel depth through cross-view feature matching and cost-volume aggregation, then generate Gaussian primitives from the resulting depth map. For example, MVSplat [11] uses plane-sweeping cost aggregation to ensure multi-view geometric consistency, enabling robust depth localization even with sparse views. DepthSplat [12] enhances this by incorporating monocular depth priors and self-supervised DINOv2 [18], [19] features, improving robustness in low-texture and occluded regions. TranSplat [13] adopts a Transformer-based, depth-aware scheme, integrating cross-view matching [20], [21] and Gaussian parameter prediction within an end-to-end framework. In contrast, direct or implicit encoders, including pixelSplat [15], latentSplat [16], and NoPoSplat [17], use probabilistic sampling, latent decoding.

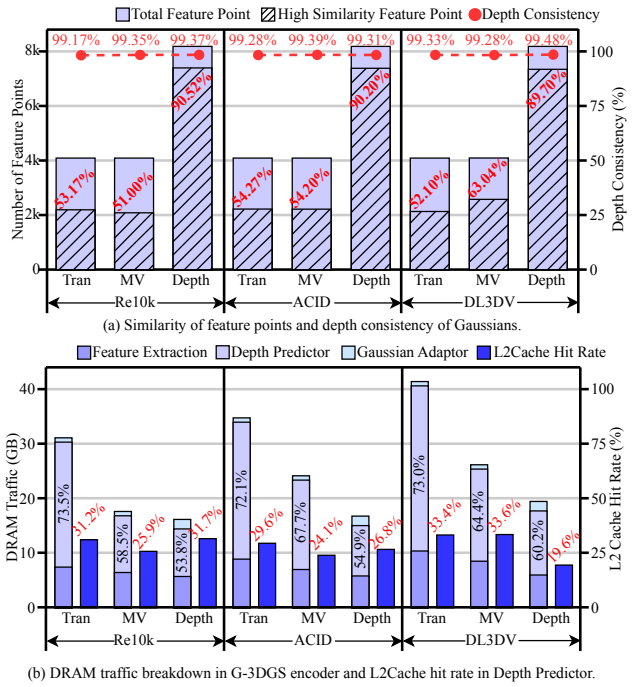
B. Depth-Guided G-3DGS Encoder Pipeline

Despite differences in backbone and matching strategy, MVSplat, DepthSplat, and TranSplat share a common execution pipeline. A CNN backbone [22], optionally enhanced with Transformer encoder layers [23], processes all input views to generate multi-scale feature maps through hierarchical convolution and self-attention. For each reference-view pixel (u, v), depth estimation is performed independently: each of the D depth hypotheses $\{d_k\}$ projects the pixel to the target view, where corresponding features are bilinearly sampled, accumulating per-hypothesis matching costs into a cost volume $\mathcal{C} \in \mathbb{R}^{H\times W\times D}$. A U-Net [24] refines $\mathcal{C}$ across depth and spatial dimensions, and a Softmax-based depth head converts the refined scores into a continuous depth estimate $\hat{d}_{u,v} = \sum_k d_k \cdot \mathrm{softmax}(-\mathcal{C}_{u,v})_k$ [25]. Finally, the predicted depth map and encoder features are regressed into per-pixel Gaussian attributes, including 3D position μ, covariance Σ, opacity α, and spherical harmonics coefficients $\mathbf{c}_{sh}$, forming the renderable Gaussian set $\mathcal{G}$ [7]. The pipeline evaluates D depths for each of the $H \times W$ pixels and emits one primitive per pixel, with no content adaptation. For one scene update and N rendered views, $T_{\mathrm{total}} = T_{\mathrm{enc}} + NT_{\mathrm{rast}}$. Our Orin NX profiling shows that one encoder pass takes about $50\times$ [11], [15] as long as one rasterized view, consistent with prior G-3DGS timing reports. The encoder thus remains the dominant cost in edge deployments where the scene representation is refreshed within roughly 50 rendered views.

C. Challenges for Accelerating Depth-Guided G-3DGS Encoders on Edge Platforms

Depth-guided, cost-volume-based G-3DGS encoders exhibit two key characteristics: **(i)** independent per-pixel depth search across D hypotheses, and **(ii)** a serial two-stage dataflow for depth prediction followed by Gaussian generation. We profiled TranSplat [13] (Tran), MVSplat [11] (MV), and DepthSplat [12] (Depth) on a Jetson Orin NX, an edge GPU with a limited 2 MB L2 cache, using scenes from the

[1]The SCARF simulator, Chisel RTL implementation, and evaluation artifact are open source at https://github.com/MAdrid1011/SCARF and archived at https://doi.org/10.5281/zenodo.22434931.



(a) Similarity of feature points and depth consistency of Gaussians.

(b) DRAM traffic breakdown in G-3DGS encoder and L2Cache hit rate in Depth Predictor.

Fig. 2. Redundant memory traffic from semantics-oblivious per-pixel depth search with cache-ineffective cross-view sampling.

Re10K [26], ACID [27], and DL3DV [28] datasets, revealing two challenges to accelerator efficiency.

a) ***Challenge 1: Independent per-pixel depth search overlooks semantic similarity in 2D feature space, causing redundant memory traffic****:* Feature maps exhibit a consistent statistical property: pixels with similar feature vectors typically belong to the same object surface and have comparable depth values, based on two priors. First, co-surface pixels share texture and illumination, resulting in highly correlated features extracted by CNNs. Second, object surfaces are geometrically smooth, causing depth to vary continuously within local regions. As shown in Fig. 2(a), pixel pairs with cosine feature similarity above 0.7 account for 51.2% of all pixels on average, with 99.5% depth consistency. For DepthSplat, which uses self-supervised DINOv2 features, this proportion rises to 91.4%. These findings indicate that semantically similar pixels provide highly predictive depth estimates.

Existing systems fail to effectively leverage semantic reuse. As shown in Fig. 2(b), the depth-search stage generates over 65% of LPDDR5 memory traffic, despite an average L2 cache hit rate of only 31.2%. This inefficiency arises from structural issues: cross-view feature sampling across D depth hypotheses produces access patterns that vary with both the depth hypothesis and the target view index, resulting in a lack of physical address locality. Conventional caches depend on address continuity rather than feature-space similarity, causing semantically similar pixels scattered throughout memory to each trigger independent full-candidate scans.

b) ***Challenge 2: The serial depth-prediction and Gaussian-generation dataflow neglects local continuity in 3D geometry, causing redundant computations****:* Existing G-3DGS encoders employ per-pixel Gaussian generation to maintain generalization, resulting in depth prediction and Gaussian generation accounting for 54% of the total execution time. To characterize redundancy inside encoder outputs, we analyze local attribute similarity among the generated Gaussian primitives. Fig. 3(b) shows the spatial distribution of attribute differences for the scene in Fig. 3(a), revealing that neighboring Gaussians are mostly similar. Figs. 3(c)–(e) plot the per-attribute similarity distributions and mark the high-similarity boundary used in our redundancy analysis. We count a Gaussian as highly similar on an attribute when the covariance or spherical-harmonic cosine similarity exceeds 0.7, or when the opacity L2 distance is below 0.2. Fig. 3(f) further shows that, across the nine model and dataset pairs, 67.8% of encoder-generated Gaussians on average satisfy all three attribute tests. Thus, adjacent pixels on the same surface produce Gaussian primitives with similar attributes.

Furthermore, using local 4×4 tiles, we merge high-similarity primitives into representative primitives, yielding a 50.9% Gaussian reduction on average. Across models and datasets, the merged Gaussian sets keep the PSNR loss below 0.1% in scene rendering, indicating that these high-similarity primitives contribute highly overlapping content to the final rendering. Thus, encoder-generated Gaussian sets contain substantial local structural redundancy. In fact, the fully optimized per-scene 3DGS reference reduces primitives by 51.3% on average, further showing the large compressible space in scene representation. However, the serial dataflow requires depth prediction for all pixels before Gaussian generation. This prevents depth prediction from observing Gaussian-primitive similarity and causes significant computational redundancy.

D. Our Approach

To address these challenges, we propose SCARF, a dataflow-co-designed accelerator that leverages semantic similarity in 2D feature space and local geometric continuity in 3D space to reduce redundant memory accesses and redundant Gaussian generation:

a) ***Feature Similarity Depth Reuse (FSDR)****:* To address Challenge 1, we propose FSDR, which replaces address-keyed lookups with semantic-signature-keyed lookups within the depth-search pipeline. Upon a hit, it narrows the candidate set to a cached-anchor neighborhood, replacing the full D-candidate scan with a guided local search, thereby eliminating the redundant feature-sampling traffic.

b) ***Scene-Adaptive Early Sparsification (SAES)****:* To address Challenge 2, we propose SAES, a dataflow that fuses depth prediction with Gaussian materialization. It breaks the strict stage boundary between depth prediction and Gaussian generation. Each tile can activate only the minimally necessary materialization path before full primitive instantiation. By skipping redundant Gaussian computation at an early stage,

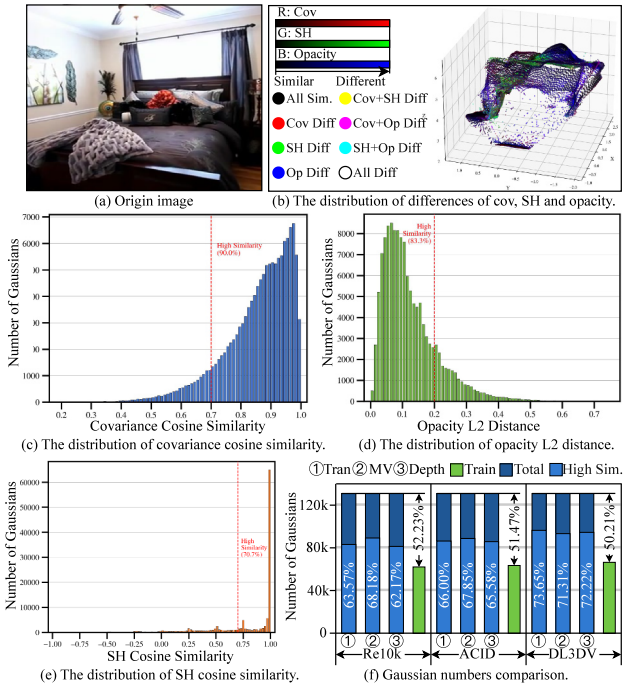


Fig. 3. Redundant Gaussian generation from the serial depth-prediction and Gaussian-generation pipeline despite local structural redundancy.

the proposed approach maintains quality and reduces encoder workload and energy consumption.

III. SCARF Dataflow

Building on FSDR and SAES, SCARF restructures the depth-guided G-3DGS encoder pipeline into four stages, as shown in Fig. 4. S1 extracts multi-view features, S2 predicts per-pixel depth, S3 materializes Gaussian primitive parameters, and S4 converts network outputs into final Gaussian primitives. Unlike a serialized S2→S3 pipeline, SCARF fuses S2 and S3 into a tile-level joint pipeline.

A. Stage 1: Feature Extraction

S1 extracts backbone features from all input views by sequentially applying CNN convolutional layers and Transformer encoder layers to produce multi-scale feature maps.

*a) **Model-agnostic runtime configuration**:* Depth-guided G-3DGS encoders vary in convolutional and Transformer layer counts and may include a Vision Transformer (ViT) branch. Memory-mapped input/output (MMIO) configuration registers manage these differences uniformly. This dataflow eliminates model-specific branches and supports depth-guided topologies on unified hardware.

*b) **Tile-aligned writeback layout**:* Feature maps at all scales are divided into fixed-size tiles before off-chip write-back. A linear layout keeps data contiguous within each tile and stores tiles sequentially, placing cross-view features of the same tile adjacent in memory. The layout matches the probe-prioritized execution order of S2, and the critical pixel reads therefore touch only a few consecutive cache lines and avoid data reordering or inter-block movement before S2.

B. Stage 2: Semantics-Guided Depth Prediction

S2 uses FSDR-guided candidate contraction to reduce memory traffic during per-pixel cross-view depth prediction.

*a) **Computation order scheduling**:* For an input tile $\mathcal{T}$, S2 constructs an ordered pixel queue $\mathcal{Q}_\mathcal{T} = [\mathcal{P}_\mathcal{T},\ \mathcal{R}_\mathcal{T}]$, where $\mathcal{P}_\mathcal{T}$ denotes the probe set used by S3 and $\mathcal{R}_\mathcal{T} = \mathcal{T} \setminus \mathcal{P}_\mathcal{T}$ contains the remaining pixels in row-major order. S2 schedules probe pixels first and places corner probes at the front of $\mathcal{P}_\mathcal{T}$. This order establishes local evidence for SAES before issuing the remaining depth searches.

*b) **Depth-candidate contraction**:* For pixel (u, v), S2 performs semantics-guided selection over $\mathcal{D} = \{d_i\}_{i=1}^{D}$. Given feature $\mathbf{x}_{u,v} \in \mathbb{R}^C$ and hyperplanes $\{\mathbf{r}_k\}_{k=1}^{K}$, S2 computes the LSH [29] signature as

$$h_k(\mathbf{x}_{u,v}) = \mathbb{I}\big(\mathbf{r}_k^\top \mathbf{x}_{u,v} \geq 0\big)\,, \qquad \mathbf{h}_{u,v} = [h_1, \ldots, h_K].$$

The signature $\mathbf{h}_{u,v}$ is queried against the FSDR cache, where the Hamming distance is calculated as

$$\delta_H(\mathbf{h}_{u,v}, \mathbf{h}_j) = \sum_{k=1}^{K} |h_k(\mathbf{x}_{u,v}) - h_k(\mathbf{x}_j)|\,.$$

Each bit records which side of a random hyperplane the feature vector falls on. The Hamming distance therefore counts hyperplane boundary crossings between two vectors and hardware-efficiently approximates their angular distance in the feature space.

If $\min_j \delta_H(\mathbf{h}_{u,v}, \mathbf{h}_j) \leq \tau_h$, FSDR retrieves the cached depth d_c and forms the local candidate set $\mathcal{D}'$ by selecting the nearest D/R depth values around d_c from the global candidate set $\mathcal{D}$. Otherwise, SCARF uses the global candidate set $\mathcal{D}$. This contraction occurs only before cost-volume construction. The retained candidates still use the original global depth values and pass through the same cross-view feature sampling, cost-volume construction, pretrained U-Net refinement, and depth-head regression. Thus, hit pixels preserve the baseline depth-estimation path, except that the cost-volume support is restricted around the cached depth. Weak semantic matches fall back to full global search. The final estimate $\hat{d}_{u,v}$ is written back with the current feature signature into the FSDR cache for reuse by subsequent pixels.

C. Stage 3: SAES Adaptive Materialization

S3 receives depth outputs from S2 and materializes Gaussian primitive attributes through the SAES three-level path hierarchy. SAES makes a tile-level path decision, but it does not replace the pretrained Gaussian adaptor. Probe pixels always use their original features and predicted depths as adaptor inputs, producing probe Gaussians through the same baseline adaptor. The decision only controls non-probe pixels. L0 and L1, corresponding to the representative and lightweight paths, approximate them from the probe Gaussians in Gaussian-parameter space, while Full keeps per-pixel adaptor execution. The hierarchy follows a first-hit progression **L0→L1→Full**,

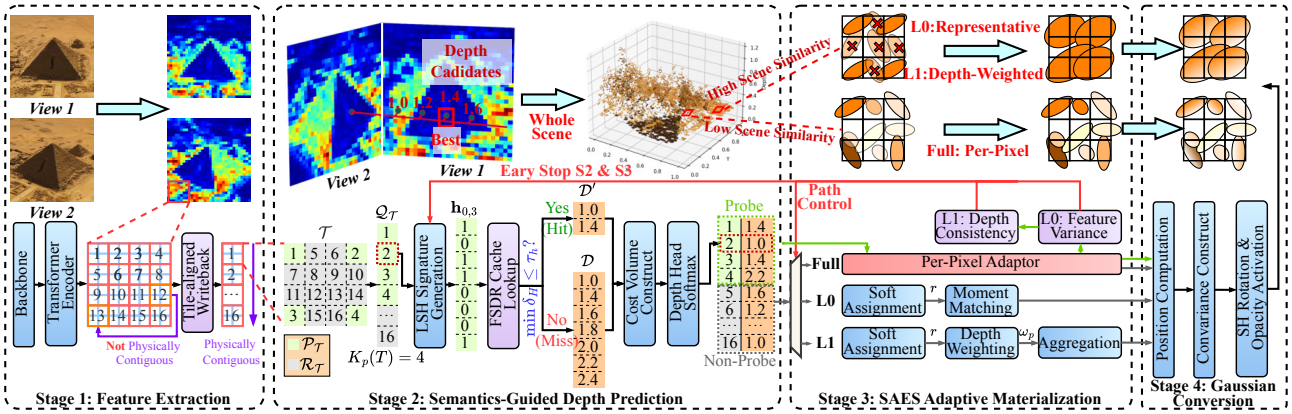


Fig. 4. SCARF dataflow overview.

where each level attempts the most aggressive pruning before falling through to the next.

*a) **Probe-set construction**:* Let the current tile be $\mathcal{T}$ with side length T pixels. The ordered probe set $\mathcal{P}_\mathcal{T} \subset \mathcal{T}$ starts from the four corners and expands slowly as tile size grows. To balance representativeness and overhead, probes are selected using a *corner-anchored and slowly expanding* strategy. For $T = 4$, only the four corners are used. For $T > 4$, the number of probes is defined as

$$K_p(T) = 4 + \left\lceil 2\log_2\left(\frac{T}{4}\right)\right\rceil, \qquad |\mathcal{P}_\mathcal{T}| = K_p(T).$$

Let $M = K_p(T) - 4$ be the number of newly added probes. The area, excluding the four corners, is divided into a $G \times G$ subgrid, where $G = \lceil\sqrt{M}\rceil$. From each subgrid, the pixel closest to the center is selected. This method prevents decision bias caused by probe clustering and ensures that the number of probes increases gradually with the tile size.

*b) **L0: feature-consistency decision**:* The L0 stage makes the earliest decision based on feature consistency among probes. Let $\bar{\mathbf{f}}$ and $\sigma^2_{\text{feat}}(\mathcal{P}_\mathcal{T})$ denote the mean and variance of the probe features $\{\mathbf{f}_p\}_{p\in\mathcal{P}_\mathcal{T}}$. If $\sigma^2_{\text{feat}}(\mathcal{P}_\mathcal{T}) < \tau_f$, the tile is classified as L0 and routed to the representative path. Low feature variance among probes indicates that the tile lies on a single-surface region. Such tiles produce Gaussians with nearly identical attributes, making probe-level representation sufficient. Only representative Gaussians from the probe path are retained. Non-probe pixels are merged into their corresponding probes via a scene-adaptive soft assignment [30]. For any pixel $i \in \mathcal{T}$, the assignment weight $r^{(\text{L0})}_{i\to p}$ combines spatial proximity and feature affinity through a joint bilateral kernel, with the spatial bandwidth modulated by the tile's own feature variance σ^2_{feat}. In smooth tiles, the spatial term defers to feature similarity. In textured tiles, assignments are tightened and localized by the spatial term:

$$r^{(\text{L0})}_{i\to p} = \frac{\exp\left(-\frac{\sigma^2_{\text{feat}}\cdot\|\mathbf{x}_i-\mathbf{x}_p\|_2^2}{\beta_x^2} - \frac{\|\mathbf{f}_i-\mathbf{f}_p\|_2^2}{\beta_f^2}\right)}{\sum_{q\in\mathcal{P}_\mathcal{T}}\exp\left(-\frac{\sigma^2_{\text{feat}}\cdot\|\mathbf{x}_i-\mathbf{x}_q\|_2^2}{\beta_x^2} - \frac{\|\mathbf{f}_i-\mathbf{f}_q\|_2^2}{\beta_f^2}\right)}.$$

Using $r^{(\text{L0})}_{i\to p}$, each probe's representative Gaussian is updated as the weighted mean μ'_p and covariance Σ'_p via first- and second-moment matching [31]. This keeps the adaptor-generated probe attributes as anchors and changes only how locally redundant pixels are absorbed. Spherical harmonics and opacity are averaged under range constraints. The L0 path offers the highest compression and is ideal for areas with smooth textures and geometry.

*c) **L1: depth-consistency decision**:* L1 is accessed only if L0 misses. At this stage, lightweight depth estimation is performed on the probes. Let $\bar{d}$ and $\sigma_d(\mathcal{P}_\mathcal{T})$ denote the mean and standard deviation of probe depths $\{d_p\}_{p\in\mathcal{P}_\mathcal{T}}$. If $\sigma_d(\mathcal{P}_\mathcal{T}) < \tau_d$, the tile is classified as L1 and directed to the lightweight path. A low depth standard deviation indicates a locally planar surface. On such surfaces, Gaussian positions and covariances vary minimally across pixels, so probe-anchored aggregation captures the dominant geometry without per-pixel instantiation. L1 retains the probe-constrained aggregation framework of L0 but introduces a depth reliability factor ω_p to recalibrate aggregation weights. This factor normalizes each probe's depth deviation $|d_p - \bar{d}|$ by the tile's own depth spread $\sigma_d(\mathcal{P}_\mathcal{T})$. Near-planar tiles tolerate little deviation, whereas depth-varying tiles have proportionally greater tolerance for deviation:

$$\omega_p = \exp\left(-\frac{|d_p - \bar{d}|}{\beta_d \cdot \sigma_d(\mathcal{P}_\mathcal{T}) + \epsilon}\right).$$

Because L1 passes depth validation, it applies less compression than L0. This makes L1 better suited for areas characterized by smooth geometry.

*d) **Full: complete computation**:* If L1 does not hit, the tile is assigned to the Full path, in which complete S2 depth search and per-pixel Gaussian adaptor execution in S3 are performed. The Full path is intended for geometrically complex regions, such as object boundaries and depth discontinuities, where preserving all primitives is necessary for rendering quality. This fallback keeps the baseline per-pixel adaptor path for tiles whose local evidence supports neither representative materialization nor lightweight materialization.

D. Stage 4: Gaussian Conversion

S4 uses the Gaussian Generation Unit (GGU) Array to convert retained depth and attributes into renderable primitives $\{\mu, \Sigma, \alpha, \mathbf{c}_{sh}\}$, then commits them to off-chip memory through two bandwidth optimizations.

a) ***Compact writeback format****:* S4 reduces output bandwidth by using a compact writeback format aligned with the Gaussian representation. It writes only the six independent upper-triangular elements of the symmetric covariance matrix and stores only the valid spherical-harmonics coefficients for the configured order. This design cuts writeback bytes per Gaussian by over 35% without altering computational semantics and ensures that output bandwidth exhibits linear scaling with scene complexity.

b) ***Streaming commit****:* During S4 execution, results are committed to the writeback buffer in tile-local Gaussian order, avoiding the burstiness and queuing overhead of conventional batch-based writeback. For tiles routed to the L0 representative or L1 lightweight path chosen by SAES, S4 receives only the retained outputs from that path, thereby further reducing conversion and writeback traffic.

IV. SCARF Architecture

The SCARF architecture (Fig. 5) comprises five interconnected components: the Matrix Vector Unit (MVU), FSDR Cache, GGU Array, control subsystem, and scratchpad memory (SPM) subsystem. The MVU shares a matrix-multiplication compute unit (MMCU) across S1 through S3, amortizing compute area across their diverse operators. The FSDR Cache intercepts per-pixel depth-search requests at the S2 front end, eliminating redundant candidate scans before they reach the MMCU. The GGU Array offloads geometric conversion from the MVU, allowing computation of the next tile while primitive construction proceeds concurrently. Together, these units keep the high-frequency dependency chain among features, depths, and Gaussians on chip, leaving only essential inputs and outputs on the DRAM path.

A. Matrix-Vector Unit

Across S1, S2, and S3, the operator mix shifts from matrix-intensive convolution and attention in S1, through irregular bilinear sampling and reduction in S2, to SAES-specific vector statistics in S3. Provisioning stage-specific arrays would fragment area and force costly data reordering at stage boundaries. SCARF instead organizes the MVU around a shared MMCU for convolution, general matrix multiplication (GEMM), and attention, augmented by a BilinearUnit for irregular cross-view sampling, a VectorALU for reductions and SAES path-selection statistics, and fused NormUnit and ActivationUnit to eliminate intermediate writebacks.

a) ***S1: feature extraction****:* The MMCU operates in Conv, GEMM, and Attention modes. In Conv mode, convolutions are mapped via *im2col* with fused normalization and activation. The NormUnit and ActivationUnit handle these operations to eliminate intermediate writeback. In GEMM mode, the same array processes QKV projections and FFN blocks. In Attention mode, the VectorALU executes scaled dot-product and normalization alongside the MMCU, enabling self-attention and cross-attention.

b) ***S2: depth prediction****:* S2 reuses the MMCU across both Conv and GEMM modes for U-Net refinement and depth-head regression, switching between them via weight-switch commands to the WeightBuffer to avoid stalls caused by array reconfiguration. A dedicated 32-way BilinearUnit absorbs this irregular traffic concurrently, keeping the MMCU fully occupied with refinement. To produce $\hat{d}_{u,v}$, the VectorALU then normalizes over depth candidates and computes the expectation over those candidates.

c) ***S3: feedback execution and Gaussian-parameter prediction****:* The single-instruction, multiple-data (SIMD) datapath of the VectorALU computes the L0 feature variance, $\sigma^2_{\text{feat}}(\mathcal{P}_\mathcal{T})$, and L1 depth standard deviation, $\sigma_d(\mathcal{P}_\mathcal{T})$, online through a reduction tree with constant path-selection latency for any probe count. In the representative-generation path, the VectorALU performs weighted moment matching to incorporate non-probe pixels into representative Gaussians. In the Full path, the MMCU Conv mode is reused for parameter prediction with fused NormUnit and ActivationUnit execution.

B. Feature Similarity Depth Reuse Cache

a) ***Storage structure****:* As illustrated in Fig. 6, the FSDR Cache uses a 32-entry CAM because its primary reuse targets, feature-similar pixels, are physically scattered across the frame. Pixels with nearly identical features and depths may have distant memory addresses, so address-domain caches miss most reuse opportunities regardless of capacity. CAM instead indexes signature values through parallel associative lookup across all entries, locating semantic matches regardless of physical address. Each 41-bit entry stores a valid bit, a 16-bit LSH signature, an FP16 depth anchor, and an 8-bit least-recently-used (LRU) counter.

b) ***Key generation and lookup****:* The LSHHashUnit projects the 128-dimensional feature vector $\mathbf{x}_{u,v}$ onto $K = 16$ random hyperplanes stored in on-chip ROM, extracting sign bits to generate the signature $\mathbf{h}_{u,v}$. Random hyperplane projection preserves the cosine-similarity structure of CNN features: pixels lying on the same surface cluster into nearby regions of signature space, so a small Hamming distance reliably predicts depth agreement. For each cache entry j, the following expression computes the distance:

$$\delta_H^{(j)} = \text{popcount}(\mathbf{h}_{u,v} \oplus \mathbf{h}_j).$$

A 32-entry parallel XOR-popcount array evaluates all entries in one cycle. A reduction tree then selects the best-matching entry j^* and its distance δ_H^*, and the cached depth d_c is read and compared to τ_h in the next cycle. Upon a hit, the Controller narrows the candidate set from $\mathcal{D}$ to $\mathcal{D}'$ centered at d_c, with size $|\mathcal{D}'| = D/R$, so that the search focuses on the most likely depth neighborhood.

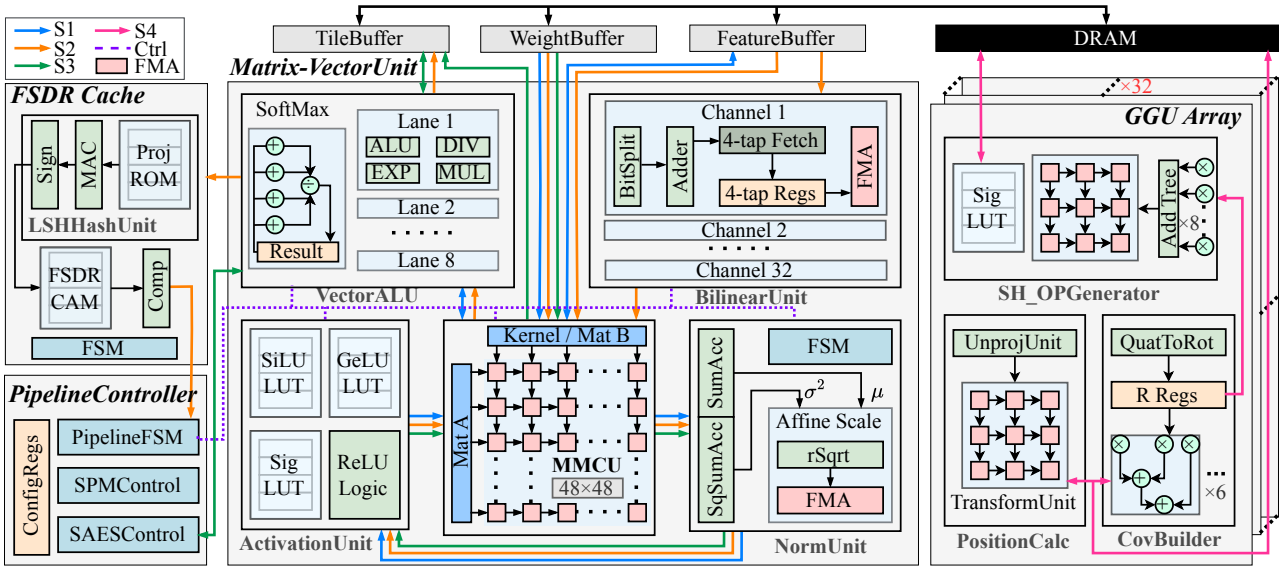


Fig. 5. SCARF architecture overview.

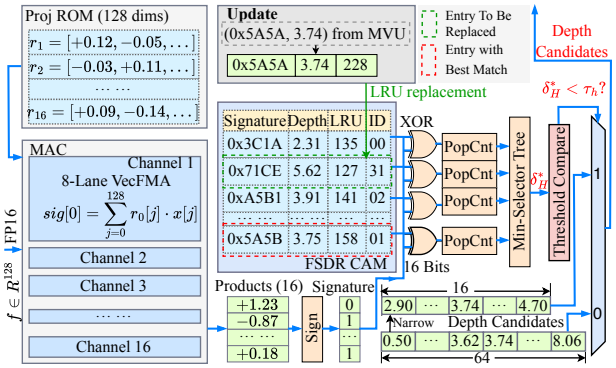


Fig. 6. FSDR cache datapath.

*c) **Update policy**:* After the current pixel completes regression and produces $\hat{d}_{u,v}$, the Controller updates the semantic-neighborhood cache by writing back $(\mathbf{h}_{u,v}, \hat{d}_{u,v})$. Empty entries are filled first. Once full, the entry with the smallest LRU counter is replaced to prioritize active semantic patterns in the current frame.

C. Gaussian Generation Unit

The GGU Array performs the final geometric mapping, converting raw per-pixel network outputs into renderable Gaussian primitives. Since these geometric operations are independent across Gaussians, the array comprises 32 processing elements, each executing a three-stage pipeline: *PositionCalc* back-projects and applies extrinsic transformations to obtain the world-space position μ. *CovBuilder* expands the quaternion into a rotation matrix R and constructs the covariance matrix $\Sigma = RSS^\top R^\top$. *SH_OPGenerator* rotates SH coefficients into world coordinates via per-band Wigner-D multiplication and applies sigmoid activation to opacity.

*a) **Rotation-matrix bypass**:* Within a GGU processing element, spherical harmonics rotation is the most memory-intensive sub-stage, comprising about 43% of total GGU memory traffic. Since the required rotation matrix R is mathematically identical to the one computed in CovBuilder, SCARF directly connects CovBuilder's R-Regs to SH_OPGenerator's input via a bypass. This eliminates the downstream stall while waiting for them to be reloaded before Wigner-D multiplication, enabling SH_OPGenerator to start immediately after CovBuilder has completed its work.

*b) **Precision-sensitive mixed-precision strategy**:* 3DGS rendering is sensitive to geometric precision: multi-stage accumulation in back-projection and covariance construction can amplify FP16 errors into visible artifacts, whereas spherical harmonics rotation is more tolerant of reduced precision. SCARF therefore applies FP32 selectively to geometry-critical steps, including PositionCalc, CovBuilder, and the rotation matrices in SH_OPGenerator, preserving geometric fidelity without the bandwidth cost of full FP32.

D. SCARF Scheduling and Model Adaptation

The algorithmic evolution of depth-guided, cost-volume-based G-3DGS encoders outpaces hardware iteration cycles. SCARF bridges this gap with a programmable architecture that enables a single hardware instance to efficiently support multiple models within the same depth-guided, cost-volume-based encoder family. Its programming model uses instruction sequences to define the computation graph, with each instruction specifying the operator type, input/output buffer addresses, and hardware configuration. At runtime, the PipelineFSM reads instructions sequentially, dispatches tasks

```
# Input from S1: FB[1], FB[1,tgt], IMG
# Stage 2: Depth Prediction
# 2a. Epipolar sampling + cost volume
BILINEAR(in=FB[1,tgt], out=TMP[0], n=64),
BILINEAR(in=FB[1], out=FB[5], scale=8),
GEMM(inA=FB[1,ref], inB=TMP[0],
  out=TMP[1], mode=dot),
REDUCE(in=TMP[1], out=FB[2],
  dim=V, op=MEAN),
# 2b. U-Net
FOR l IN unet:
 CONV(in=FB[2], w=W[unet,l], out=FB[2]),
 NORM(in=FB[2], out=FB[2], type=BN),
 ACTIVATION(in=FB[2], out=FB[2], type=RELU),
# 2c. Depth head
CONV(in=FB[2], w=W[depth_head],
  out=FB[3], k=1),
ACTIVATION(in=FB[3],
  out=FB[3], type=SOFTMAX),
GEMM(inA=FB[3],
  inB=DRAM[depth_hyp], out=TB[0]),
```

```
# Stage 3: Gaussian Parameter Prediction
CONCAT(in=[FB[5],TB[0]], out=FB[5]),
# 3a. Refinement convolutions
FOR l IN refine:
 CONV(in=FB[5], w=W[refine,l], out=FB[5]),
 NORM(in=FB[5], out=FB[5], type=BN),
 ACTIVATION(in=FB[5], out=FB[5], type=RELU),
# 3b. Parameter heads
# scales, rotations, SH
CONV(in=FB[5], w=W[gaussian_head],
  out=FB[6], k=1),
# refined depth
CONV(in=FB[5], w=W[depth_refine],
  out=FB[7], k=1),
# disparity + opacity
CONV(in=FB[5], w=W[disparity_head],
  out=FB[8], k=1),
STORE(in=FB[8:6], out=TB[3:1])
```

(a) TranSplat Core Instruction Flow

```
# Input from S1: FB[cnn], FB[mv],
# FB[mono_vit], IMG
# Stage 2: Depth Prediction
# 2a. Multi-view matching + cost volume
BILINEAR(in=FB[mv,tgt], out=TMP[0], n=128),
GEMM(inA=FB[mv,ref], inB=TMP[0],
  out=TMP[1], mode=dot),
REDUCE(in=TMP[1], out=FB[cost],
  dim=V, op=MEAN),
# 2b. CV + CNN + ViT + mono-ViT regressor
CONCAT(in=[FB[cost],FB[cnn],FB[mv],
  FB[mono_vit]], out=FB[reg_in]),
FOR l IN reg:
 CONV(in=FB[reg_in], w=W[reg,l], out=FB[reg]),
 NORM(in=FB[reg], out=FB[reg], type=GN),
 ACTIVATION(in=FB[reg],
   out=FB[reg], type=GELU),
CONV(in=FB[reg_in], w=W[residual], out=FB[res]),
REDUCE(in=[FB[reg],FB[res]],
  out=FB[reg], op=SUM),
# 2c. Depth head + learned DPT upsample
CONV(in=FB[reg], w=W[depth_head],
  out=FB[prob], k=3),
ACTIVATION(in=FB[prob],
  out=FB[prob], type=SOFTMAX),
GEMM(inA=FB[prob],
  inB=DRAM[depth_hyp], out=TB[low_d]),
BILINEAR(in=TB[low_d], out=TB[bilin_d], scale=4),
CONV(in=DPT[mono,cnn,mv,low_d],
  w=W[dpt_depth], out=TB[res_d]),
REDUCE(in=[TB[bilin_d],TB[res_d]],
  out=TB[depth], op=SUM),
```

```
# Stage 3: Gaussian Parameter Prediction
# 3a. Feature DPT + match probability
CONV(in=DPT[mono,cnn,mv],
  w=W[dpt_feat], out=FB[up_feat]),
REDUCE(in=FB[prob],
  out=TB[match], dim=D, op=MAX),
BILINEAR(in=TB[match],
  out=TB[match], mode=nearest),
# 3b. Gaussian regressor
CONCAT(in=[IMG,TB[depth],
  TB[match],FB[up_feat]], out=FB[g_in]),
CONV(in=FB[g_in], w=W[g_reg0],
  out=FB[g_mid], k=3),
ACTIVATION(in=FB[g_mid],
  out=FB[g_mid], type=GELU),
CONV(in=FB[g_mid], w=W[g_reg1],
  out=FB[g_reg], k=3),
# 3c. Gaussian head
CONCAT(in=[FB[g_reg],IMG,
  FB[up_feat],TB[match]], out=FB[h_in]),
CONV(in=FB[h_in], w=W[g_head0],
  out=FB[h_mid], k=3),
ACTIVATION(in=FB[h_mid],
  out=FB[h_mid], type=GELU),
CONV(in=FB[h_mid], w=W[g_head1],
  out=TB[raw_g], k=3),
STORE(in=[TB[depth],TB[match],
  TB[raw_g]], out=TB[gaussian])
```

(b) DepthSplat Core Instruction Flow

Fig. 7. S2/S3 instructions for TranSplat and DepthSplat.

to hardware units based on the operator type, and manages data movement among the buffers.

SCARF defines six hardware primitives: `CONV` for convolution and deconvolution, `GEMM` for matrix multiplication, `NORM` for BatchNorm and LayerNorm, `ACTIVATION` for ReLU, GELU, and Softmax, `BILINEAR` for bilinear sampling, and `REDUCE` for Mean, Max, and Sum. Higher-level operators such as Transformer attention and epipolar sampling are decomposed into these primitives. This abstraction maintains a stable hardware base while adapting to model variations through instruction scheduling. Fig. 7 compares the S2 and S3 instructions of TranSplat and DepthSplat. The comparison exposes two concrete schedule differences. First, DepthSplat carries the mono ViT feature branch from S1 into S2, where cost-volume refinement fuses CNN, multi-view, and mono features, while TranSplat refines the epipolar cost volume and regresses depth. Second, DepthSplat adds DPT-based upsampling before Gaussian prediction, whereas the lighter depth and feature outputs go directly into the Gaussian heads that belong to TranSplat.

V. Methodology

*a) **Experimental Platform**:* We implemented the SCARF RTL design using the TSMC 28 nm process and the OpenROAD physical-design flow, including synthesis, placement, routing, and post-layout timing analysis [32]. On-chip SRAM blocks were generated using a commercial memory compiler within the same technology node. The design operates at 1 GHz and features a fully pipelined architecture. To evaluate SCARF under realistic workloads, we developed a cycle-accurate simulator that models the SCARF microarchitecture and integrates Ramulator [33] to capture detailed DRAM timing for Micron LPDDR4X-4266. The simulator performs both functional and timing simulations, accepts multi-view images as input, and outputs predicted 3D Gaussian parameters. Performance metrics are derived from simulator cycle counts. Logic power is estimated from the post-layout gate-level netlist, TSMC 28 nm Liberty timing and power files, and workload switching activity generated by the cycle-accurate simulator. The switching activity is annotated in the OpenROAD and OpenSTA analysis flow to estimate dynamic power and leakage power. On-chip SRAM energy is estimated from simulator access counts and memory-compiler energy numbers. Using DRAM activity traces generated by the cycle-accurate simulator, DRAMPower [34] provides the corresponding off-chip memory energy estimate.

*b) **Workloads**:* We selected three representative depth-guided, cost-volume-based G-3DGS models: MVSplat [11] (MV), TranSplat [13] (Tran), and DepthSplat [12] (Depth), which differ in architecture and computational characteristics. MVSplat is the lightest, TranSplat has medium complexity, and DepthSplat is the most compute-intensive due to its heavier ViT backbone and larger depth search space. The datasets used include Re10K [26], ACID [27], and DL3DV [28], covering diverse indoor/outdoor, driving, and dense multi-view scenes. All models use official pretrained weights.

*c) **Baseline**:* Previous accelerators have primarily focused on per-scene NeRF rendering [35], [36] and 3DGS rasterization [37], [38], but none have targeted feed-forward G-3DGS encoder construction. We use the Jetson Orin NX 16 GB as the system-level baseline, a common edge AI platform in robotics and autonomous driving. It features an Ampere GPU with 1,024 CUDA cores, 16 GB of LPDDR5 memory at 102.4 GB/s, and a 2 MB L2 cache, running at 918 MHz in MAXN mode (25 W TDP). Encoder and rasterizer latencies were measured separately using Nsight Systems. The optimized software stack employs PyTorch [39] with cuDNN [40], cuBLAS, optimized epipolar sampling, cost-volume and attention paths, and custom CUDA kernels for model-specific operators. The implementation uses FP16 precision and CUDA Graphs throughout.

VI. Evaluation

A. Performance and Quality

*a) **End-to-End Performance**:* Fig. 8(a) summarizes the end-to-end performance of SCARF ASIC over Orin NX. Across nine model and dataset pairs, SCARF reaches a geometric-mean speedup of 2.94× and an area-normalized throughput of 0.168 inf/s/mm^2, which is 16.1× higher than Orin NX. The gain comes from algorithm-level work reduction and hardware mapping. In S2, FSDR uses semantic hashing and cache lookup to select a local depth window, reducing

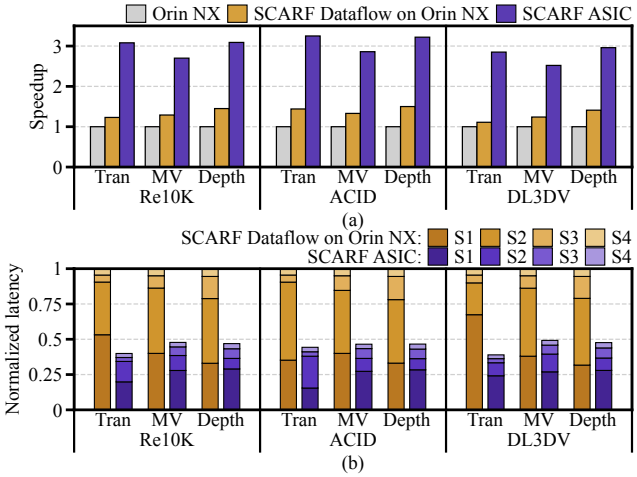


Fig. 8. End-to-end speedup of SCARF.

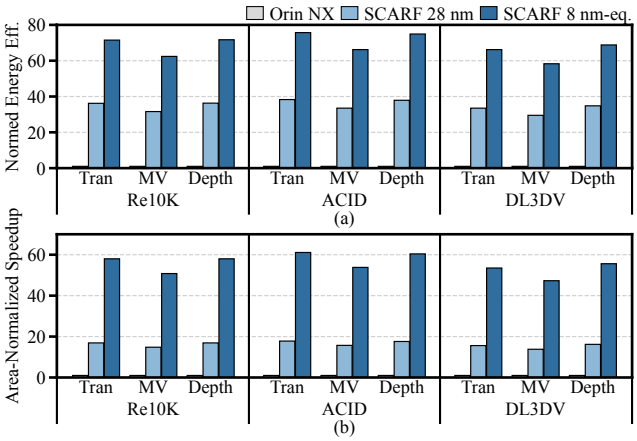


Fig. 9. Area efficiency and energy efficiency of SCARF.

depth-candidate search and feature-buffer accesses. At the tile level, SAES identifies local redundancy and routes selected tiles to the representative or lightweight path, reducing full Gaussian materialization. To separate the contribution of the dataflow from ASIC mapping, we implement SCARF Dataflow on Orin NX. This configuration keeps the Orin NX GPU backend unchanged, adds FSDR depth-window selection and SAES tile-level path selection to the encoder. It reaches a geometric-mean speedup of 1.33×, showing that FSDR and SAES also reduce useful work on the GPU. The GPU gain remains smaller because hashing, path selection, compaction, and small-kernel scheduling lower GPU efficiency.

*b) **Latency Breakdown**:* Fig. 8(b) decomposes the latency of SCARF Dataflow on Orin NX and SCARF ASIC into S1 through S4. The dataflow-on-GPU stack is normalized to 1 for each model and dataset pair. The ASIC stack uses the same baseline, so each colored segment shows the remaining latency of the corresponding stage after hardware mapping. The ASIC stacks occupy 0.39 to 0.49 of the normalized dataflow height. S2 contributes the largest shrinkage. It reaches a geometric-mean speedup of 3.96× because FSDR contracts the depth-search space from D to $D/4$ and reduces feature-buffer accesses. S3 reaches a geometric-mean speedup of 1.80× because SAES reduces tiles that require full per-pixel Gaussian materialization. S1 and S4 reach geometric-mean speedups of 1.62× and 1.54×. S1 improves because its convolution and attention layers map to long dense MMCU kernels with fused normalization and activation, avoiding small-kernel launch and intermediate-writeback overheads in the GPU baseline. S4 improves because the GGU maps per-Gaussian conversion operations onto a dedicated pipeline, while SAES reduces the number of Gaussians entering S4.

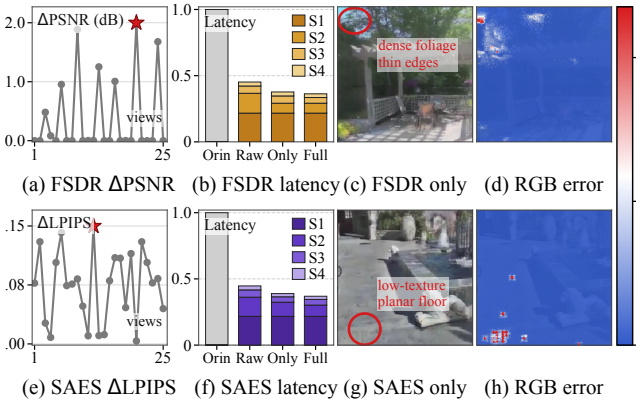
*c) **Energy Efficiency**:* Figure 9 compares the measured 28 nm results of SCARF with 8 nm-equivalent results estimated by DeepScaleTool [41]. The 8 nm-equivalent results control for the process-node gap between SCARF and Orin NX, so they primarily reflect the contribution of architecture and dataflow. Across multiple model and dataset pairs, SCARF reaches a geometric-mean energy-efficiency improvement of 68.2× and an area-normalized throughput improvement of 55.2× under this process-normalized view. This gain comes from work reduction and specialized hardware mapping. FSDR and SAES reduce depth search, feature-buffer traffic, and Gaussian materialization, so each inference performs fewer high-energy accesses and redundant computations. Under the less advanced 28 nm process, the measured post-layout implementation still reaches 34.5× and 16.1×. Concretely, the process gap reduces the energy-efficiency gain by 1.98× on average and the area-normalized throughput gain by 3.43× on average. Therefore, the architecture and dataflow design of the G-3DGS encoder account mainly for the efficiency advantage achieved by the SCARF design.

*d) **Overall Quality**:* Table I shows that SCARF stays close to the baseline across all nine pairs. The maximum changes are 0.072% in PSNR, 0.24% in SSIM, and 0.002 in LPIPS, with no consistent degradation. Quality is preserved by the adaptive mechanisms in both approximation stages. FSDR's local candidate window bounds depth deviations and confines DL3DV fluctuations to $\leq$0.19% PSNR at abrupt depth boundaries. SAES's scene-adaptive $r_{i\to p}$ and ω_p suppress over-smoothing at textured boundaries and relax in uniform regions. For example, TranSplat on ACID gains 0.07% PSNR and -0.001 LPIPS. Where metrics diverge, the shifts reflect localized artifacts rather than global fidelity loss.

*e) **Worst-Case Error Analysis**:* Fig. 10 shows view-wise quality losses, normalized latency breakdowns, and RGB error maps for the worst FSDR and SAES views. Fig. 10(a) reports FSDR-only ΔPSNR across all evaluated views, and Fig. 10(e) reports SAES-only ΔLPIPS. The red stars mark the worst views shown in Fig. 10(c) and (g). For FSDR, the error remains localized because the full-candidate fallback avoids narrowing depth search when semantic matching is unreliable. The remaining gain comes from restricting reliable S2 matches to local depth windows and reducing feature-buffer accesses. For SAES, the worst LPIPS case concentrates errors

TABLE I
SCARF RENDERING QUALITY VERSUS BASELINE.

Model	Dataset	PSNR↑ Base / SCARF	SSIM↑ Base / SCARF	LPIPS↓ Base / SCARF
Tran	Re10K	28.08 / 28.07	0.919 / 0.917	0.128 / 0.130
	ACID	29.58 / 29.60	0.899 / 0.901	0.145 / 0.144
	DL3DV	25.08 / 25.07	0.854 / 0.852	0.178 / 0.180
MV	Re10K	30.22 / 30.23	0.942 / 0.944	0.128 / 0.123
	ACID	32.08 / 32.06	0.916 / 0.915	0.144 / 0.146
	DL3DV	28.02 / 28.05	0.914 / 0.912	0.145 / 0.146
Depth	Re10K	28.43 / 28.42	0.925 / 0.923	0.122 / 0.123
	ACID	29.43 / 29.44	0.910 / 0.910	0.137 / 0.137
	DL3DV	26.93 / 26.98	0.885 / 0.886	0.157 / 0.156



(a) FSDR ΔPSNR (b) FSDR latency (c) FSDR only (d) RGB error

(e) SAES ΔLPIPS (f) SAES latency (g) SAES only (h) RGB error

Fig. 10. Worst-case view losses, latency and error maps.

at sparse local structures where representative or lightweight materialization changes Gaussian attributes. These local errors are more visible to LPIPS because LPIPS is sensitive to boundaries, texture, and shadow. In Table I, the largest LPIPS increase is only from 0.178 to 0.180, indicating no persistent degradation in global quality from the local worst-view errors.

B. Source of Performance Gains

a) ***Speedup Ablation****:* To assess the independent impact of the two core optimizations on the same hardware, we report ablation results relative to the SCARF (no optimization) baseline. This baseline retains the SCARF compute array, on-chip buffers, and control framework but disables FSDR and SAES, restoring the original serialized S2→S3 execution flow, effectively serving as a vanilla accelerator. Fig. 11 isolates the performance gains of FSDR and SAES. We evaluate three configurations: FSDR only, SAES only, and both enabled. FSDR alone yields a geometric mean speedup of 1.35× by contracting the depth-search space. SAES alone achieves a 1.26× speedup via tile-level path selection. Combined, they reach a 1.59× speedup indicating largely independent, additive benefits. The strongest synergy occurs on ACID (1.70×) and the weakest on DL3DV (1.49×), which matches the dataset-level trends reported above.

b) ***FSDR Detailed Analysis****:* Table II breaks down the mechanisms behind the 1.35× FSDR contribution. The *Guided*

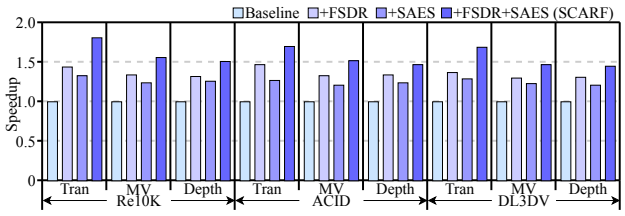


Fig. 11. Ablation study of FSDR and SAES.

TABLE II
FSDR PER-MODEL STATISTICS.

Model	Dataset	Guided Rate	Top-1 Cov.	Depth Evals Saved in S2 (M)	Feat. Buf. Red. (MB)
Tran	Re10K	72.1%	99.91%	4.54 (54.2%)	415 (51.5%)
	ACID	76.0%	99.93%	4.78 (57.4%)	438 (54.4%)
	DL3DV	65.0%	99.88%	4.09 (48.9%)	374 (46.5%)
MV	Re10K	66.2%	99.90%	1.04 (49.6%)	95 (47.2%)
	ACID	71.0%	99.87%	1.12 (53.4%)	102 (50.7%)
	DL3DV	58.0%	99.83%	0.91 (43.4%)	84 (41.7%)
Depth	Re10K	83.0%	99.89%	5.22 (62.2%)	478 (59.4%)
	ACID	87.0%	99.95%	5.47 (65.2%)	501 (62.3%)
	DL3DV	78.0%	99.85%	4.91 (58.5%)	449 (55.8%)

Rate is the fraction of pixels with narrowed search spaces via cached depth anchors, which ranges from 58% to 87%. As the Guided Rate increases, *Depth Evals Saved in S2* rises from 43% to 65%, while *Feat. Buf. Red.* rises from 41% to 62%, indicating that the FSDR gain comes from reducing depth-candidate evaluation and LPDDR4X feature-buffer traffic. *Top-1 Cov.* measures the fraction of guided local depth windows that still cover the top-1 candidate selected by full search. It remains between 99.83% and 99.95% across all model and dataset pairs. Thus, almost all full-search top-1 candidates remain inside the local windows, consistent with the stronger semantic consistency of DINOv2 features on roads, sky, and building planes. MVSplat has the lowest Guided Rate on DL3DV at 58%, because lightweight features form less stable matches under complex occlusion and repeated textures. Its Top-1 Cov. still reaches 99.83%, showing that complex scenes mainly reduce the triggering scope of guided search rather than shifting the triggered windows away from full-search results.

c) ***SAES Detailed Analysis****:* Table III similarly traces SAES's 1.26× contribution to tile-level path selection statistics. The *L0 Rate* indicates the fraction of tiles entering the representative-generation path from 12.0% to 46.7%, while the *L1 Rate* shows the fraction entering the lightweight path from 8.5% to 25.1%. Together, these rates determine the workload bypassing the Full path, leading to *Gaussians Saved* from 16.1% to 33.8% and *S2 Evals Saved* from 0.45M to 3.92M by skipping S2 search for L0 or L1 tiles. *Low-Var. Agree.* measures the fraction of L0 or L1 tiles whose full per-pixel materialization already has low-variation Gaussian attributes. It remains between 99.23% and 100.00% across all combinations, showing that SAES mainly compresses Gaussian outputs that are already locally consistent under full materialization.

TABLE III
SAES PER-MODEL STATISTICS.

Model	Dataset	L0 Rate	L1 Rate	Low-Var. Agree.	Gaussians Saved	S2 Evals Saved (M)
Tran	Re10K	17.2%	14.8%	100.00%	20.2%	1.45
	ACID	21.3%	11.2%	99.86%	24.3%	1.79
	DL3DV	12.0%	10.1%	99.23%	17.0%	1.01
MV	Re10K	29.0%	12.5%	100.00%	24.0%	0.61
	ACID	36.0%	18.0%	99.77%	28.0%	0.76
	DL3DV	21.3%	8.5%	99.75%	16.1%	0.45
Depth	Re10K	38.7%	18.6%	99.59%	29.2%	3.25
	ACID	46.7%	25.1%	99.85%	33.8%	3.92
	DL3DV	30.7%	13.2%	99.62%	24.8%	2.57

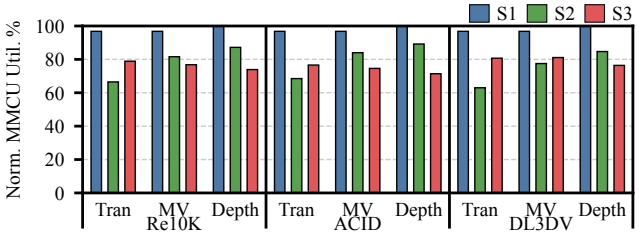


Fig. 12. Normalized stage-wise MMCU active utilization.

On DL3DV, complex textures and depth boundaries keep more tiles on the Full path, reducing TranSplat's L0 Rate to 12.0%. Its Low-Var. Agree. still reaches 99.23%, indicating that complex regions mainly reduce early-path triggering rather than causing early paths to cover many unsuitable tiles.

*d) **MMCU Utilization**:* Fig. 12 reports normalized MMCU active utilization for S1, S2, and S3. S1 reaches 97.9% utilization, making backbone feature extraction a stable high-load stage. S2 averages 78.0% because depth prediction mixes matrix refinement with cost-volume sampling, FSDR-guided search, and candidate reduction. TranSplat interleaves more sampling and reduction between MMCU kernels, MVSplat uses a lighter cost volume, and DepthSplat exposes denser mono-ViT refinement. S3 averages 76.7% and follows SAES path selection. Representative and lightweight tiles reduce full Gaussian-head execution, so DepthSplat is lower on Re10K and ACID. S4 runs on the GGU Array and overlaps with the preceding three stages, requiring no additional MMCU operations. The shared MMCU therefore remains active across S1, S2, and S3 and preserves the area amortization benefit provided by the shared substrate.

C. Sensitivity Study

*a) **FSDR Cache Size**:* Fig. 13 relates FSDR cache size to throughput, access overhead, tile working-set coverage, and miss rate. Each cache entry stores one feature signature and its depth anchor, and the cache is reset at the beginning of each frame. Tile working-set coverage is the fraction of guided pixels whose reusable depth anchor is already resident when probed. From 8 to 128 entries, coverage rises while capacity miss falls. Both trends saturate by 32 entries, where geometric-mean throughput also peaks. Capacity miss drops by 1.48 percentage points from 16 to 32 entries, but only by 0.22 and

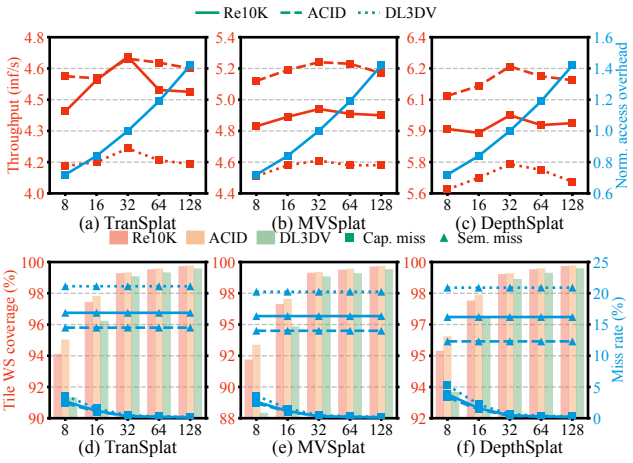


Fig. 13. Sensitivity to FSDR cache size.

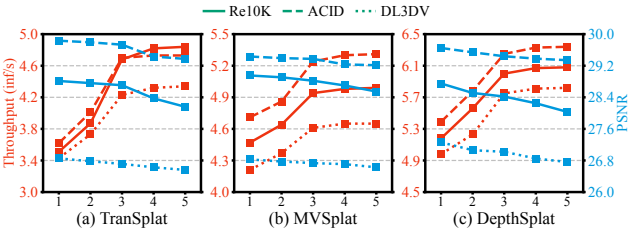


Fig. 14. Sensitivity to FSDR Hamming threshold τ_h.

0.12 points over the next two doublings. At the same time, normalized access overhead rises from 1.00 at 32 entries to 1.19 and 1.42 at 64 and 128 entries, because a larger CAM raises per-lookup cost. Past 32 entries, the extra capacity gives little miss reduction, so access overhead becomes the main limiter and throughput declines slightly at 64 and 128 entries. Fig. 13(d)–(f) further shows that semantic miss stays near 26.07% at all sizes, indicating that residual misses are set by scene content and feature similarity. Per-frame reset also has limited runtime impact. In our measurements, filling 32 distinct anchors takes 130 guided-pixel probes on average, only 0.198% of a 256×256 view. Overall, 32 entries cover the main tile working set. Smaller caches increase capacity miss, while larger caches mainly increase lookup overhead, so we use 32 entries as the default.

*b) **FSDR Hamming Threshold**:* Fig. 14 illustrates the effect of the FSDR Hamming threshold, τ_h, on system performance. τ_h sets how similar two pixel features must be before FSDR uses a cached depth to narrow the search range. Since this decision is local, scene changes mainly alter the number of pixels that satisfy the match condition, and therefore the FSDR hit rate and speedup. The threshold choice depends instead on the marginal reliability of the extra matches admitted as Hamming distance grows. Larger τ_h first adds stable near-feature matches, then weaker matches with smaller throughput gains and higher quality risk. The same trend appears across

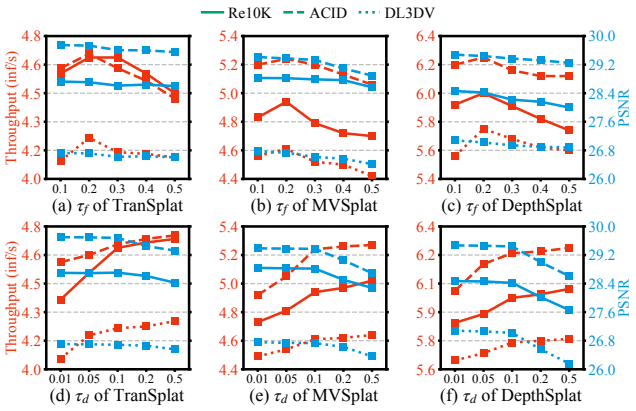


Fig. 15. Sensitivity to SAES variance thresholds τ_f and τ_d.

Re10K, ACID, and DL3DV, indicating that the transition from reliable reuse to weak matching does not shift strongly with scene distribution. Increasing τ_h from 1 to 3 raises geometric-mean throughput by about 18.1%, while increasing it to 5 adds only 1.5%, with a 0.20% gain from 4 to 5. Quality fluctuation remains below 1.15% at $\tau_h = 3$, but grows to 3.47% at $\tau_h = 5$. We therefore set $\tau_h = 3$ by default to capture reliable reuse and avoid weak matches.

*c) **SAES Variance Thresholds**:* Fig. 15(a)–(c) evaluates the feature-variance threshold, τ_f, while panels (d)–(f) assess the depth-variance threshold, τ_d. Both thresholds are derived from probe statistics inside each tile. They measure local appearance consistency and local geometry consistency, tying the decision to tile-local image evidence rather than to a scene category or model type. Model and scene changes alter the prevalence of these local patterns, and therefore change the L0 and L1 rates and speedup. They do not significantly change the relation between variance and local approximability. The measurements follow this behavior. τ_f shows a consistent unimodal trend across all nine combinations and peaks at 0.2. Further relaxing τ_f routes more appearance-varying tiles to L0, increasing moment-matching overhead and quality fluctuation. For τ_d, throughput saturates beyond 0.1, while quality drops sharply at $\tau_d \geq 0.2$. This indicates that the main locally smooth geometry has already been captured by L1 and that further relaxation mainly admits tiles with stronger depth variation. We therefore use $\tau_f = 0.20$ and $\tau_d = 0.1$ as defaults. They mark the boundary where local approximability changes from reliable to unreliable.

*d) **Tile Size**:* Fig. 16 illustrates the impact of tile size on the effectiveness of SAES. Throughput peaks at a 4×4 tile size across all nine model-dataset pairs. Both smaller and larger tiles reduce performance. The geometric mean throughput decreases by approximately 4.1% for 2×2 tiles and by 9.6%, 12.1%, and 12.7% for 8×8, 16×16, and 32×32 tiles, respectively, compared to 4×4. Smaller tiles improve local decision granularity but increase scheduling and path-switching overhead, offsetting computational gains. Larger

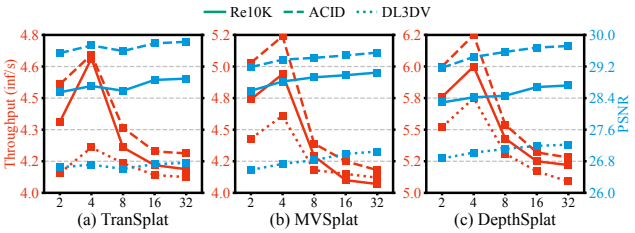


Fig. 16. Sensitivity to SAES tile size.

tiles excessively average local texture and geometry, reducing L0/L1 early stopping triggers and causing more regions to revert to the Full path, thereby decreasing pruning efficiency. Quality generally improves with tile size. The 32×32 setting yields the highest quality, only 1.67% better than 4×4, but incurs over a 12% throughput loss. Balancing quality and performance, 4×4 is selected as the default tile size.

D. Area and Power Analysis

*a) **Area Analysis**:* As shown in Table IV, SCARF occupies 27.8 mm^2 in 28 nm, with the area distribution directly reflecting the workload hierarchy. The MMCU dominates at 50.4% (14.0 mm^2) because it serves as the shared compute substrate across S1 through S3. Concentrating compute into one 48×48 array avoids the duplication of stage-specific designs. The GGU Array accounts for 17.3% (4.80 mm^2), sized to run its 32 geometry-conversion PEs concurrently with the MVU. The FSDR subsystem occupies only 0.08 mm^2 (0.3%), including only its 32-entry CAM, confirming that semantic indexing imposes negligible silicon cost relative to the 41% to 62% DRAM traffic reduction it enables. On-chip buffers (1.26 mm^2, 4.5%) sustain inter-stage data reuse, while the LPDDR4X PHY (3.50 mm^2, 12.6%) reflects the unavoidable fixed analog cost of high-speed I/O.

*b) **Power Analysis**:* In Table IV, SCARF consumes 2.1 W at 1 GHz and 1.0 V, enabling fanless operation. Dynamic power mirrors the area hierarchy: the MMCU dominates at 52.2% (1,016 mW) as the primary active-compute fabric, while I/O and PHY account for 12.5% (244.0 mW) as the fixed cost of off-chip bandwidth. Despite consuming only 15 mW (<1%), the FSDR subsystem delivers outsized savings. By cutting DRAM traffic by 41% to 62%, it suppresses the I/O switching energy that would otherwise grow with every redundant depth-search cycle. SAES compounds this effect through tile-level path pruning, reducing S3 dynamic power proportionally to the Gaussian reduction in Table III.

VII. Related Work

*a) **Feed-forward 3D reconstruction and 3DGS rendering accelerators**:* Recent accelerators target feed-forward 3D reconstruction and 3DGS rendering workloads. VersaQ-3D [42] accelerates VGGT-based generalizable 3D reconstruction through versatile quantization and a reconfigurable datapath for transformer inference. GCC [37] optimizes the

TABLE IV
SCARF AREA AND POWER BREAKDOWN IN 28 NM.

Component	Area (mm^2)	Dyn. (mW)	Stat. (mW)
MVU (15.35 mm^2, 1,133.6 / 82.3 mW)			
MMCU	14.0 (50.4%)	1,016 (52.2%)	73.8 (50.9%)
VectorALU	0.50 (1.8%)	46.7 (2.4%)	2.1 (1.4%)
BilinearUnit	0.82 (3.0%)	61.9 (3.2%)	4.8 (3.3%)
NormUnit	0.02 (0.1%)	5.8 (0.30%)	0.9 (0.62%)
ActivationUnit	0.01 (0.04%)	3.2 (0.16%)	0.7 (0.48%)
GGU Array (×32 PEs) (4.80 mm^2, 314.6 / 25.6 mW)			
PositionCalc	0.65 (2.3%)	42.1 (2.2%)	3.4 (2.3%)
CovBuilder	1.35 (4.9%)	89.3 (4.6%)	7.2 (5.0%)
SH_OPGenerator	2.80 (10.1%)	183.2 (9.4%)	15.0 (10.3%)
FSDR Subsystem (0.08 mm^2, 15.3 / 1.2 mW)			
LSHHashUnit	0.03 (0.1%)	6.1 (0.31%)	0.5 (0.34%)
CAM Array (32-entry)	0.04 (0.1%)	7.8 (0.40%)	0.5 (0.34%)
FSDR Controller	0.01 (0.04%)	1.4 (0.07%)	0.2 (0.14%)
On-chip Buffers (1.26 mm^2, 153.3 / 18.7 mW)			
Weight Buffer (128 KB)	0.38 (1.4%)	41.2 (2.1%)	6.3 (4.3%)
Feature Buffer (256 KB)	0.70 (2.5%)	89.4 (4.6%)	10.1 (7.0%)
Tile Buffer (64 KB)	0.18 (0.6%)	22.7 (1.2%)	2.3 (1.6%)
Control & Clock (0.47 mm^2, 86.2 / 5.2 mW)			
Control + Interconnect	0.32 (1.2%)	27.8 (1.4%)	1.4 (1.0%)
PLL + Clock tree	0.15 (0.5%)	58.4 (3.0%)	3.8 (2.6%)
I/O & PHY (3.50 mm^2, 244.0 / 12.0 mW)			
I/O + LPDDR4X PHY	3.50 (12.6%)	244.0 (12.5%)	12.0 (8.3%)
Routing / filler (85% util.)	2.32 (8.3%)	—	—
Total die	**27.8 mm^2**	**2,092 mW**	

coupling between preprocessing and rendering with cross-stage conditional processing and Gaussian-wise rendering, reducing unnecessary preprocessing and repeated Gaussian loading. Neo [43] targets real-time on-device 3DGS rendering and exploits temporal redundancy in depth ordering across consecutive frames through reuse-and-update sorting. These works optimize either transformer-based reconstruction or rendering from an existing Gaussian set. SCARF targets depth-guided, cost-volume-based G-3DGS encoder inference, where cost-volume depth search and Gaussian materialization dominate encoder construction. By reducing this encoder-side work before rendering, SCARF sits upstream of rasterization and complements GCC and Neo.

*b) **3DGS memory scaling and training systems**:* A separate line of systems work addresses memory capacity limits in large-scale 3DGS reconstruction and training. CLM [44] offloads Gaussian parameters to CPU memory and loads the active working set into GPU memory when needed, using pipelining and access-pattern-aware communication reduction to support very large scenes on a single consumer GPU. GS-Scale [45] similarly stores Gaussian parameters, gradients, and optimizer states in host memory, then reduces training overhead with selective offloading, parameter forwarding, and deferred optimizer updates. These systems expand the scale of per-scene 3DGS optimization and rendering, but they do not change the computation inside feed-forward G-3DGS encoders. SCARF targets this different stage by reducing cost-volume depth search and redundant Gaussian materialization before the final Gaussian set is emitted.

VIII. CONCLUSION

SCARF targets inference in depth-guided G-3DGS encoders with cost-volume depth prediction. It exploits semantic locality through Feature Similarity Depth Reuse, which narrows depth candidates by feature-space affinity, and local geometric redundancy through Scene-Adaptive Early Sparsification, which activates the minimal materialization path before full primitive instantiation. Across three depth-guided, cost-volume-based models and three datasets, SCARF delivers a 2.94× geometric-mean end-to-end speedup, 2.23 inferences per joule, and a maximum PSNR drop under 0.072%. Implemented in 28 nm, SCARF occupies 27.8 mm^2 and consumes 2.1 W, demonstrating practical potential for edge novel-view synthesis.

For downstream integration, SCARF serves as the Gaussian producer before a rasterizer. S4 writes primitives to a buffer that an existing 3DGS rasterizer reads through its input interface. For an end-to-end G-3DGS pipeline, future work can study cross-stage scheduling, double buffering, and on-chip reuse at this boundary.

REFERENCES

[1] H. Huang, L. Li, H. Cheng, and S. Yeung, "Photo-SLAM: Real-time simultaneous localization and photorealistic mapping for monocular, stereo, and RGB-D cameras," in *IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR 2024, Seattle, WA, USA, June 16-22, 2024*. IEEE, 2024, pp. 21 584–21 593. [Online]. Available: https://doi.org/10.1109/CVPR52733.2024.02039

[2] H. Matsuki, R. Murai, P. H. J. Kelly, and A. J. Davison, "Gaussian splatting SLAM," in *2024 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2024, pp. 18 039–18 048.

[3] Z. Yang, Y. Chen, J. Wang, S. Manivasagam, W.-C. Ma, A. J. Yang, and R. Urtasun, "UniSim: A neural closed-loop sensor simulator," in *2023 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2023, pp. 1389–1399.

[4] A. Yu, V. Ye, M. Tancik, and A. Kanazawa, "pixelNeRF: Neural radiance fields from one or few images," in *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2021, pp. 4576–4585.

[5] Q. Wang, Z. Wang, K. Genova, P. Srinivasan, H. Zhou, J. T. Barron, R. Martin-Brualla, N. Snavely, and T. Funkhouser, "IBRNet: Learning multi-view image-based rendering," in *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2021, pp. 4688–4697.

[6] A. Chen, Z. Xu, F. Zhao, X. Zhang, F. Xiang, J. Yu, and H. Su, "MVSNeRF: Fast generalizable radiance field reconstruction from multi-view stereo," in *2021 IEEE/CVF International Conference on Computer Vision (ICCV)*, 2021, pp. 14 104–14 113.

[7] B. Kerbl, G. Kopanas, T. Leimkuehler, and G. Drettakis, "3D gaussian splatting for real-time radiance field rendering," *ACM Trans. Graph.*, vol. 42, no. 4, Jul. 2023. [Online]. Available: https://doi.org/10.1145/3592433

[8] Z. Yu, A. Chen, B. Huang, T. Sattler, and A. Geiger, "Mip-Splatting: Alias-free 3D gaussian splatting," in *2024 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2024, pp. 19 447–19 456.

[9] T. Lu, M. Yu, L. Xu, Y. Xiangli, L. Wang, D. Lin, and B. Dai, "Scaffold-GS: Structured 3D gaussians for view-adaptive rendering," in *2024 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2024, pp. 20 654–20 664.

[10] J. L. Schönberger and J.-M. Frahm, "Structure-from-Motion revisited," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 4104–4113.

[11] Y. Chen, H. Xu, C. Zheng, B. Zhuang, M. Pollefeys, A. Geiger, T.-J. Cham, and J. Cai, "MVSplat: Efficient 3D gaussian splatting from sparse multi-view images," in *Computer Vision – ECCV 2024*, A. Leonardis, E. Ricci, S. Roth, O. Russakovsky, T. Sattler, and G. Varol, Eds. Cham: Springer Nature Switzerland, 2025, pp. 370–386.

[12] H. Xu, S. Peng, F. Wang, H. Blum, D. Barath, A. Geiger, and M. Pollefeys, "DepthSplat: Connecting gaussian splatting and depth," in *2025 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2025, pp. 16 453–16 463.

[13] C. Zhang, Y. Zou, Z. Li, M. Yi, and H. Wang, "TranSplat: generalizable 3D gaussian splatting from sparse multi-view images with transformers," in *Proceedings of the Thirty-Ninth AAAI Conference on Artificial Intelligence and Thirty-Seventh Conference on Innovative Applications of Artificial Intelligence and Fifteenth Symposium on Educational Advances in Artificial Intelligence*, ser. AAAI'25/IAAI'25/EAAI'25. AAAI Press, 2025. [Online]. Available: https://doi.org/10.1609/aaai.v39i9.33070

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

[15] D. Charatan, S. L. Li, A. Tagliasacchi, and V. Sitzmann, "pixelSplat: 3D gaussian splats from image pairs for scalable generalizable 3D reconstruction," in *IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR 2024, Seattle, WA, USA, June 16-22, 2024*. IEEE, 2024, pp. 19 457–19 467. [Online]. Available: https://doi.org/10.1109/CVPR52733.2024.01840

[16] C. Wewer, K. Raj, E. Ilg, B. Schiele, and J. E. Lenssen, "latentSplat: Autoencoding variational gaussians for fast generalizable 3D reconstruction," in *Computer Vision – ECCV 2024*, A. Leonardis, E. Ricci, S. Roth, O. Russakovsky, T. Sattler, and G. Varol, Eds. Cham: Springer Nature Switzerland, 2025, pp. 456–473.

[17] B. Ye, S. Liu, H. Xu, X. Li, M. Pollefeys, M.-H. Yang, and S. Peng, "No pose, no problem: Surprisingly simple 3D gaussian splats from sparse unposed images," in *International Conference on Learning Representations*, Y. Yue, A. Garg, N. Peng, F. Sha, and R. Yu, Eds., vol. 2025, 2025, pp. 54 009–54 033. [Online]. Available: https://proceedings.iclr.cc/paper_files/paper/2025/file/857b34d81f0a8bfe3e18879dee3b5086-Paper-Conference.pdf

[18] M. Caron, H. Touvron, I. Misra, H. Jegou, J. Mairal, P. Bojanowski, and A. Joulin, "Emerging properties in self-supervised vision transformers," in *2021 IEEE/CVF International Conference on Computer Vision (ICCV)*, 2021, pp. 9630–9640.

[19] M. Oquab, T. Darcet, T. Moutakanni, H. V. Vo, M. Szafraniec, V. Khalidov, P. Fernandez, D. Haziza, F. Massa, A. El-Nouby, M. Assran, N. Ballas, W. Galuba, R. Howes, P. Huang, S. Li, I. Misra, M. Rabbat, V. Sharma, G. Synnaeve, H. Xu, H. Jégou, J. Mairal, P. Labatut, A. Joulin, and P. Bojanowski, "DINOv2: Learning robust visual features without supervision," *Trans. Mach. Learn. Res.*, vol. 2024, 2024. [Online]. Available: https://openreview.net/forum?id=a68SUt6zFt

[20] Y. Ding, W. Yuan, Q. Zhu, H. Zhang, X. Liu, Y. Wang, and X. Liu, "TransMVSNet: Global context-aware multi-view stereo network with transformers," in *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2022, pp. 8575–8584.

[21] J. Sun, Z. Shen, Y. Wang, H. Bao, and X. Zhou, "LoFTR: Detector-free local feature matching with transformers," in *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2021, virtual, June 19-25, 2021*. Computer Vision Foundation / IEEE, 2021, pp. 8922–8931. [Online]. Available: https://openaccess.thecvf.com/content/CVPR2021/html/Sun_LoFTR_Detector-Free_Local_Feature_Matching_With_Transformers_CVPR_2021_paper.html

[22] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016*. IEEE Computer Society, 2016, pp. 770–778. [Online]. Available: https://doi.org/10.1109/CVPR.2016.90

[23] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, J. Uszkoreit, and N. Houlsby, "An image is worth 16x16 words: Transformers for image recognition at scale," in *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net, 2021. [Online]. Available: https://openreview.net/forum?id=YicbFdNTTy

[24] O. Ronneberger, P. Fischer, and T. Brox, "U-Net: Convolutional networks for biomedical image segmentation," in *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2015*, N. Navab, J. Hornegger, W. M. Wells, and A. F. Frangi, Eds. Cham: Springer International Publishing, 2015, pp. 234–241.

[25] A. Kendall, H. Martirosyan, S. Dasgupta, P. Henry, R. Kennedy, A. Bachrach, and A. Bry, "End-to-end learning of geometry and context for deep stereo regression," in *2017 IEEE International Conference on Computer Vision (ICCV)*, 2017, pp. 66–75.

[26] T. Zhou, R. Tucker, J. Flynn, G. Fyffe, and N. Snavely, "Stereo magnification: learning view synthesis using multiplane images," *ACM Trans. Graph.*, vol. 37, no. 4, Jul. 2018. [Online]. Available: https://doi.org/10.1145/3197517.3201323

[27] A. Liu, A. Makadia, R. Tucker, N. Snavely, V. Jampani, and A. Kanazawa, "Infinite nature: Perpetual view generation of natural scenes from a single image," in *2021 IEEE/CVF International Conference on Computer Vision (ICCV)*, 2021, pp. 14 438–14 447.

[28] L. Ling, Y. Sheng, Z. Tu, W. Zhao, C. Xin, K. Wan, L. Yu, Q. Guo, Z. Yu, Y. Lu, X. Li, X. Sun, R. Ashok, A. Mukherjee, H. Kang, X. Kong, G. Hua, T. Zhang, B. Benes, and A. Bera, "DL3DV-10K: A large-scale scene dataset for deep learning-based 3d vision," in *IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR 2024, Seattle, WA, USA, June 16-22, 2024*. IEEE, 2024, pp. 22 160–22 169. [Online]. Available: https://doi.org/10.1109/CVPR52733.2024.02092

[29] M. S. Charikar, "Similarity estimation techniques from rounding algorithms," in *Proceedings of the Thirty-Fourth Annual ACM Symposium on Theory of Computing*, ser. STOC '02. New York, NY, USA: Association for Computing Machinery, 2002, p. 380–388. [Online]. Available: https://doi.org/10.1145/509907.509965

[30] D. Comaniciu and P. Meer, "Mean shift: A robust approach toward feature space analysis," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 24, no. 5, p. 603–619, May 2002. [Online]. Available: https://doi.org/10.1109/34.1000236

[31] Z. Fan, K. Wang, K. Wen, Z. Zhu, D. Xu, and Z. Wang, "LightGaussian: Unbounded 3D gaussian compression with 15x reduction and 200+ FPS," in *Advances in Neural Information Processing Systems*, A. Globerson, L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. Tomczak, and C. Zhang, Eds., vol. 37. Curran Associates, Inc., 2024, pp. 140 138–140 158. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2024/file/fd881d3b625437354d4421818f81058f-Paper-Conference.pdf

[32] T. Ajayi, V. A. Chhabria, M. Fogaça, S. Hashemi, A. Hosny, A. B. Kahng, M. Kim, J. Lee, U. Mallappa, M. Neseem, G. Pradipta, S. Reda, M. Saligane, S. S. Sapatnekar, C. Sechen, M. Shalan, W. Swartz, L. Wang, Z. Wang, M. Woo, and B. Xu, "Toward an open-source digital flow: First learnings from the OpenROAD project," in *Proceedings of the 56th Annual Design Automation Conference 2019*, ser. DAC '19. New York, NY, USA: Association for Computing Machinery, 2019. [Online]. Available: https://doi.org/10.1145/3316781.3326334

[33] H. Luo, Y. C. Tuğrul, F. N. Bostancı, A. Olgun, A. G. Yağlıkçı, and O. Mutlu, "Ramulator 2.0: A modern, modular, and extensible DRAM simulator," *IEEE Comput. Archit. Lett.*, vol. 23, no. 1, p. 112–116, Jan. 2024. [Online]. Available: https://doi.org/10.1109/LCA.2023.3333759

[34] L. Steiner, T. Psota, M. Mörz, D. Christ, M. Jung, and N. Wehn, "DRAMPower 5: An open-source power simulator for current generation DRAM standards," in *Proceedings of the Rapid Simulation and Performance Evaluation for Design Workshop*, ser. RAPIDO '25. New York, NY, USA: Association for Computing Machinery, 2025, p. 8–16. [Online]. Available: https://doi.org/10.1145/3721848.3721850

[35] S.-H. Noh, B. Shin, J. Choi, S. Lee, J. Kung, and Y. Kim, "FlexNeRFer: A multi-dataflow, adaptive sparsity-aware accelerator for on-device NeRF rendering," in *Proceedings of the 52nd Annual International Symposium on Computer Architecture*, ser. ISCA '25. New York, NY, USA: Association for Computing Machinery, 2025, p. 1894–1909. [Online]. Available: https://doi.org/10.1145/3695053.3731107

[36] S. Li, C. Li, W. Zhu, B. T. Yu, Y. K. Zhao, C. Wan, H. You, H. Shi, and Y. C. Lin, "Instant-3D: Instant neural radiance field training towards on-device AR/VR 3D reconstruction," in *Proceedings of the 50th Annual International Symposium on Computer Architecture*, ser. ISCA '23. New York, NY, USA: Association for Computing Machinery, 2023. [Online]. Available: https://doi.org/10.1145/3579371.3589115

[37] M. Pei, G. Li, J. Si, Z. Zhu, Z. Mo, P. Wang, Z. Song, X. Liang, and J. Cheng, *GCC: A 3DGS Inference Architecture with Gaussian-Wise and Cross-Stage Conditional Processing*. New York, NY, USA: Association for Computing Machinery, 2025, p. 1824–1837. [Online]. Available: https://doi.org/10.1145/3725843.3756072

[38] J. Lee, S. Lee, J. Lee, J. Park, and J. Sim, "GSCore: Efficient radiance field rendering via architectural support for 3D gaussian splatting," in *Proceedings of the 29th ACM International Conference*

on Architectural Support for Programming Languages and Operating Systems, Volume 3, ser. ASPLOS '24. New York, NY, USA: Association for Computing Machinery, 2024, p. 497–511. [Online]. Available: https://doi.org/10.1145/3620666.3651385

[39] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Köpf, E. Z. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "PyTorch: An imperative style, high-performance deep learning library," in *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, H. M. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. B. Fox, and R. Garnett, Eds., 2019, pp. 8024–8035. [Online]. Available: https://proceedings.neurips.cc/paper/2019/hash/bdbca288fee7f92f2bfa9f7012727740-Abstract.html

[40] S. Chetlur, C. Woolley, P. Vandermersch, J. Cohen, J. Tran, B. Catanzaro, and E. Shelhamer, "cuDNN: Efficient primitives for deep learning," *CoRR*, vol. abs/1410.0759, 2014. [Online]. Available: http://arxiv.org/abs/1410.0759

[41] S. Sarangi and B. M. Baas, "DeepScaleTool: A tool for the accurate estimation of technology scaling in the deep-submicron era," in *2021 IEEE International Symposium on Circuits and Systems (ISCAS)*, 2021, pp. 1–5.

[42] Y. Zhang, J. Cheng, X. Liu, Z. Li, C. J. Li, J. Wu, L. Jiang, Y. Xie, J. Xu, and W. Zhang, "VersaQ-3D: A reconfigurable accelerator enabling feed-forward and generalizable 3D reconstruction via versatile quantization," 2026. [Online]. Available: https://arxiv.org/abs/2601.20317

[43] C. Oh, S. Oh, J. Hwang, Y. Kim, H. Sharma, and J. Park, "Neo: Real-time on-device 3D gaussian splatting with reuse-and-update sorting acceleration," in *Proceedings of the 31st ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*, ser. ASPLOS 2026. New York, NY, USA: Association for Computing Machinery, 2026, pp. 1268–1284. [Online]. Available: https://doi.org/10.1145/3779212.3790192

[44] H. Zhao, X. Min, X. Liu, M. Gong, Y. Li, A. Li, S. Xie, J. Li, and A. Panda, "CLM: Removing the GPU memory barrier for 3D gaussian splatting," in *Proceedings of the 31st ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*, ser. ASPLOS 2026. New York, NY, USA: Association for Computing Machinery, 2026, pp. 377–393. [Online]. Available: https://doi.org/10.1145/3779212.3790140

[45] D. Lee, D. Jeong, J. W. Lee, and H. Yoon, "GS-Scale: Unlocking large-scale 3D gaussian splatting training via host offloading," in *Proceedings of the 31st ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*, ser. ASPLOS 2026. New York, NY, USA: Association for Computing Machinery, 2026, pp. 860–875. [Online]. Available: https://doi.org/10.1145/3779212.3790167

APPENDIX
ARTIFACT APPENDIX

A. Abstract

The SCARF artifact was awarded the Artifacts Available, Artifacts Evaluated (Functional), and Results Reproduced badges. The v1.0.4 artifact provides the Python functional and cycle simulators, Feature Similarity Depth Reuse, Scene-Adaptive Early Sparsification, adapters for TranSplat, MVSplat, and DepthSplat, Chisel RTL, workflow automation, result validators, and a public physical-design proxy. The release maps Table 1, Figure 8, Figure 11, and their supporting records to reproducible commands and machine-readable acceptance rules. The evaluated scope is stated in Section G.

B. Artifact check-list (meta-information)

- **Algorithm:** Feature Similarity Depth Reuse and Scene-Adaptive Early Sparsification for semantic depth reuse and pre-materialization Gaussian pruning.
- **Program:** Python simulator, Chisel RTL, Docker, and Bash automation covering the released accelerator datapath, control, and memory system used in evaluation.
- **Compilation:** Python 3.10 and CUDA 12.1. RTL checks use JDK 11+, sbt 1.9+, Chisel 6.6, and Verilator 5+.
- **Binary:** No precompiled binary is required. The Docker image is built locally from the supplied Dockerfile.
- **Models:** TranSplat, MVSplat, and DepthSplat.
- **Data sets:** Re10K, ACID, and DL3DV for key-result evaluation, plus the packaged synthetic fixture for installation validation.
- **Runtime environment:** Ubuntu Linux or the supplied CUDA container. Separate classic and DepthSplat profiles isolate their distinct PyTorch and CUDA dependencies.
- **Hardware:** An NVIDIA GPU with at least 8 GB VRAM for the quick workflow, 12 GB or more for quality and mechanism workflows, and a Jetson Orin NX 16 GB in MAXN mode with locked clocks for independent Figure 8 measurements.
- **Metrics:** PSNR, SSIM, LPIPS, latency, speedup, energy, and source-bound RTL cycle counts.
- **Output:** JSON records and provenance manifests that bind the source revision, inputs, commands, and output hashes, together with validator reports and generated figures.
- **Experiments:** Installation, quality, performance, mechanism ablation, and sensitivity workflows.
- **Storage and time:** Approximately 20 GB and 20–60 minutes for the first container build and quick setup. Full evaluation requires the official data sets and several GPU-hours.
- **License:** Apache-2.0 for the artifact code. Third-party data and checkpoints remain under their upstream terms.
- **Archived:** https://doi.org/10.5281/zenodo.22434931.

C. Description

1) How to access: Download the v1.0.4 source archive from https://doi.org/10.5281/zenodo.22434931, or clone https://github.com/MAdrid1011/SCARF with recursive submodules. The archive includes the quick fixture, Dockerfile, model adapters, RTL, manifests, frozen mechanism configuration, and reviewer workflows. Official data sets and model checkpoints are acquired separately by the supplied scripts under their upstream terms, then recorded in the generated input manifests. These manifests preserve the data set and checkpoint identities used during evaluation and bind them to the reviewer workflows described below.

2) Models and data: The supported model adapters are TranSplat, MVSplat, and DepthSplat. The checkpoint manifest selects the model-compatible weights for each data set. The following commands prepare the standard sources:

```
bash data/download_checkpoints.sh \
  --profile all
bash data/download_re10k.sh
```

The scripts record the selected source and place the resulting files in the paths used by the adapters. DL3DV preparation and its evaluator terms are documented in `data/download_dl3dv.sh` and `artifact/CALIBRATION.md`. Data sets and checkpoints are not redistributed in the archive.

3) Hardware and software: The quick workflow uses an x86-64 Linux host, Docker with NVIDIA support, and a CUDA 12.1-compatible GPU. Quality and mechanism workflows run the three model and data set pairs serially and benefit from at least 12 GB VRAM. Figure 8 independent measurements use a Jetson Orin NX 16 GB in MAXN mode. The public RTL path uses JDK 11+, sbt 1.9+, Chisel 6.6, and Verilator 5+ for elaboration, linting, and simulation.

D. Installation and quick workflow

The artifact-root commands below build the pinned Functional container and record its validation output:

```
docker build -t scarf-ae:1.0.4 .
mkdir -p outputs/docker-quick
docker run --rm --gpus all \
  -v "$PWD/outputs/docker-quick:/results" \
  scarf-ae:1.0.4
```

The container builds the synthetic fixture, obtains the quick checkpoint when needed, and writes a structured record under `/results` for provenance and subsequent validation.

E. Reviewer workflows

The frozen, calibrated mechanism configuration is released under `artifact/calibration/frozen/`. It lets reviewers begin quality evaluation with model execution without recalibration. The source-bound RTL timing manifest binds each cycle record to the released RTL and exported model workload evaluated during AE.

The unified reviewer plan can run all requested targets. Select `proxy` when a Jetson Orin NX is unavailable:

```
python scripts/run_reviewer_claims.py \
  --run-config artifact/key_results_run.json \
  --figure8-device proxy \
  --proxy-input \
  /path/to/reviewer-gpu-proxy.json \
  --output-root outputs/ae
```

For direct workflows, run quality first, then source-bound timing, mechanisms, and performance:

```
bash scripts/run_ae.sh quality \
  --output-root outputs/ae
bash scripts/run_ae.sh mechanisms \
  --claim-timing-manifest \
  outputs/timing-backend/manifest.json \
  --output-root outputs/ae
bash scripts/run_ae.sh performance \
  --device orin \
  --claim-timing-manifest \
  outputs/timing-backend/manifest.json \
  --output-root outputs/ae
```

The RTL timing and export scripts generate source-bound timing from exported workloads. The runner binds packed tensors to the released RTL and runs to completion without an implicit cycle ceiling. The `--max-cycles` argument is diagnostic only. The path records positive cycles for ASIC, FSDR, SAES, combined, and zero-input runs, with provenance bound to released inputs and RTL.

F. Evaluation and expected results

Table 1 reports PSNR, SSIM, and LPIPS for every committed model–data set pair, and Figure 8 reports nine latency pairs and their geometric-mean speedups. Figure 11 reports no-optimization, FSDR-only, SAES-only, and combined cycle paths with reuse-rate, utilization, and event counters. For each target, `artifact/evaluation_catalog.json` records the command, required inputs, evidence class, and acceptance rule enforced by this validator:

```
python scripts/validate_ae.py \
  --input outputs/ae --require-key-results
```

Quality tolerances are 0.15 dB PSNR, 0.005 SSIM, and 0.005 LPIPS. Figure 8 and Figure 11 geometric means allow 5% relative variation. Mechanism rates and utilization allow two percentage points, while event counts allow 5% relative variation against the frozen reference records.

G. Hardware proxy and scope

Reviewers without Orin hardware can use the portability proxy after capturing same-workload CUDA timings. The proxy reports labeled latency, FPS, and speedup estimates without independent Orin power or temperature evidence. The released public hardware path routes the RTL with iFlow and ASAP7, then reports the technology-normalized estimate defined by the paper, with raw and scaled fields kept separate. Commercial TSMC 28 nm and LPDDR collateral are outside the archive. The Results Reproduced badge applies to the evidence evaluated during AE within these stated scope boundaries, not to measurements that depend on proprietary technology files outside the archive.