SPACECACHE: INTERMEDIATE FEATURE REUSE FOR EFFICIENT SEMANTIC SEGMENTATION ON POINT CLOUD STREAMS

Anonymous authors
Paper under double-blind review

ABSTRACT

Point-cloud streams are central to autonomous driving and robotics, yet existing segmentation networks process every frame from scratch, although consecutive frames largely overlap. We observe that the intermediate features of point-cloud transformers remain semantically consistent across frames for the same spatial region, especially in deep stages. Building on this, we present **SpaceCache**, a training-free feature cache that reuses these features across frames. Tokens of previously observed regions skip a sequence of transformer blocks and take their cached features, only the remaining tokens are computed, and the full token layout is restored for dense prediction. Lightweight cache management keeps reused features reliable by limiting each entry's age and evicting low-confidence entries. Unlike token merging, SpaceCache selects tokens at negligible cost and runs its cache operations once per encoder stage. On both outdoor and indoor benchmarks, such as nuScenes, Waymo, and ScanNet, with two transformer backbones of Utonia and LitePT-B, SpaceCache achieves up to 1.65× speedup with less than 1-point mIoU drop and consistently outperforms token-merging methods.

1 INTRODUCTION

Point-cloud segmentation is a key component in real-time systems such as autonomous vehicles and robots, where low-latency inference is critical (Lang et al., 2019; Milioto et al., 2019). Much of the recent progress in this task has been driven by transformer-based backbones (Zhao et al., 2021; Wu et al., 2024; Yue et al., 2026), which treat individual points or local groups of points as tokens. However, a single LiDAR sweep often incorporates tens to hundreds of thousands of points (Sun et al., 2020; Caesar et al., 2020; Behley et al., 2019), causing these backbones to process an extremely large number of tokens per frame. This cost is further compounded in point-cloud *streams*, where such token-heavy inference must be repeated at every timestep. Despite this, accelerating inference in streaming point-cloud perception remains relatively underexplored.

In recent years, several studies have explored reducing inference latency in semantic segmentation, primarily for *single*-frame point cloud. For example, token-merging methods (Bolya et al., 2023; Norouzi et al., 2024; Tran et al., 2024; 2025) identify and merge similar, redundant tokens, thereby reducing the computation that would otherwise be performed on them. However, token merging requires similarity estimation and irregular aggregation operations such as `gather/reduce` to construct a new token sequence from the merged representations, introducing non-negligible GPU overhead (Lu et al., 2025). More recently, methods such as GitMerge3D (Tran et al., 2025) derive token similarities from layer-specific features, requiring merge and unmerge operations at every layer and causing this overhead to scale with network depth. Other approaches reduce computation through token pruning, which naturally avoids the repeated feature aggregation. Token Halting (Ye et al., 2023), for example, progressively prunes intermediate tokens but uses an additional learned token-selection module.

Notably, temporal redundancy across *multi*-frame point clouds has received relatively limited attention. One such approach, EVE (Peng et al., 2021), prunes redundant input points using lightweight geometric proximity across frames and reuses their previous decoder outputs, but this inevitably reduces the contextual information available to the remaining tokens. As a result, EVE also re-

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.

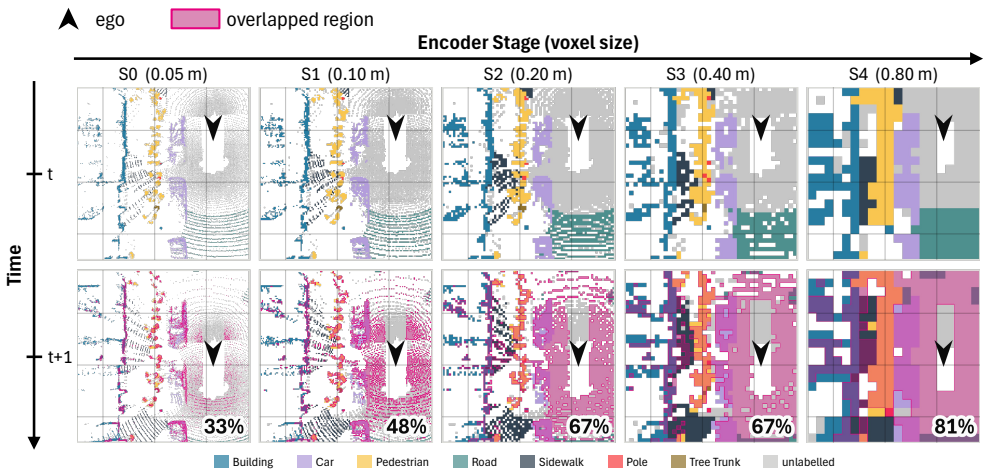


Figure 1: **Increasing temporal overlap across encoder stages.** Visualization on the Waymo dataset using the stage-wise downsampling ratios of the Utonia and LitePT-B models. Both have five encoder stages (S0–S4). Each grid cell represents the voxel region corresponding to a single token, and its color indicates the ground-truth semantic class of that region. Magenta highlights voxels in frame $t{+}1$ that overlap with frame t. As progressive downsampling increases the voxel size at deeper stages, tokens represent larger spatial regions and exhibit higher cross-frame overlap.

lies on an additional learned module to compensate for the resulting errors. Consequently, existing approaches either incur non-negligible overhead in token merging or rely on additional training to mitigate the contextual information loss caused by token pruning.

In this paper, we introduce **SpaceCache**, an efficient intermediate-feature-reuse technique that skips computation for spatial regions observed in earlier frames and instead reuses their cached features. By reusing *intermediate* token representations, namely, those at deeper encoder stages, our method achieves the low overhead of token pruning while mitigating its contextual information loss in a training-free manner. This design is motivated by two observations: at deeper encoder stages, progressive downsampling leads to higher temporal token overlap (Figure 1), while cached features computed in earlier frames under different contexts become increasingly semantically consistent with their corresponding current-frame features (Sec. 2.2). SpaceCache targets only a few deeper encoder stages, where much of the encoder computation is concentrated, enabling substantial latency reductions. Moreover, by caching features once per stage rather than at every layer within a stage, it keeps the cache footprint to only a few MB. Our key contributions are as follows:

- **Temporal redundancy grows with depth.** We show that deeper encoder stages exhibit higher cross-frame token overlap and feature consistency (Sec. 2.2). This observation motivates SpaceCache, a training-free cache that lets tokens of previously observed regions bypass these stages.
- **Reliability-aware cache management.** Cached features can become stale on moving objects or unreliable at ambiguous regions, degrading accuracy when they are reused. We address this by combining confidence-based commit and eviction with lifetime-based eviction (Sec. 3.3).
- **Better accuracy-efficiency trade-off.** Across three benchmarks in both outdoor and indoor (nuScenes, Waymo, ScanNet) and two transformer backbones (Utonia, LitePT-B), SpaceCache consistently achieves a better accuracy-efficiency trade-off than token-merging methods, reaching up to 1.65× speedup within a 1-point mIoU drop (Sec. 4.1).

2 Background and Preliminary Study

2.1 Network Property

Point-cloud encoder architectures. Modern point-cloud encoders, such as Utonia (Zhang et al., 2026) and LitePT (Yue et al., 2026), typically adopt a hierarchical multi-stage architecture, as illus-

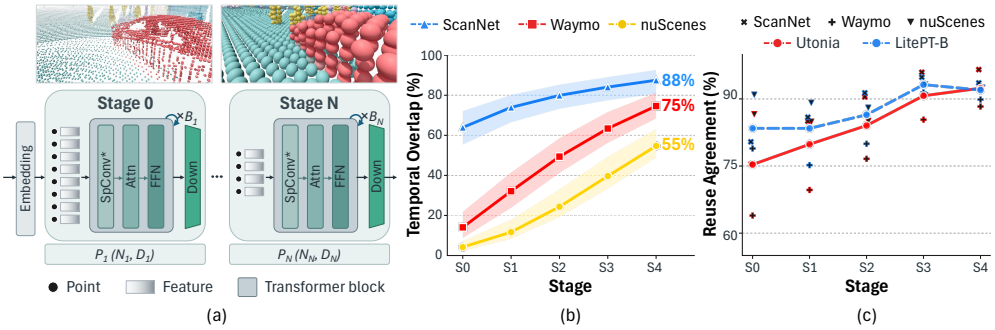


Figure 2: **Temporal-reuse opportunity in point-cloud encoders.** (a) A multi-stage encoder, where stage l runs B_l blocks over a fixed token set and downsampling between stages shrinks the token set while enlarging the region each token covers. (b) The fraction of stage-l overlapping tokens, whose region already appeared in the previous frame, grows substantially with depth. (c) The Reuse Agreement of stage-l overlapping tokens, whether the overlapped previous-frame token and the corresponding current-frame token predict the same class (Appendix A.2), also increases with depth.

trated in Figure 2(a). Each stage consists of a downsampling layer that aggregates multiple spatially nearby tokens into a single token, followed by B_l transformer blocks that may include sparse convolution, attention, and MLP operations. An input embedding layer at the beginning of the network transforms the input point coordinates into high-dimensional feature vectors.

Token coordinates and features. We refer to each point or spatially aggregated representation processed by the encoder as a *token*. Each token is associated with a three-dimensional coordinate and a high-dimensional feature vector, which together form the input to the encoder blocks. At stage l, we denote the token coordinates of block b by $P_l^{(b)}=\{p_i^{(b)}\}_{i=1}^{N_l}$, where N_l is the number of tokens and $p_i^{(b)} \in \mathbb{R}^3$ is the coordinate of token i. The output features are similarly denoted by $F_l^{(b)}=\{f_i^{(b)}\}_{i=1}^{N_l}$, where $f_i^{(b)} \in \mathbb{R}^{D_l}$ is the corresponding feature.

Consistent token coordinates within a stage. The token coordinate set changes only at downsampling layers. Since the transformer blocks within a stage preserve token coordinates, they all share the same coordinate set: $P_l^{(0)}=P_l^{(1)}=\cdots=P_l^{(B_l)}=P_l$. Consequently, once tokens from different frames are matched to the same spatial region at a given stage, this correspondence remains valid throughout all blocks in that stage. This property enables intermediate features to be reused across the entire sequence of blocks, rather than only at an individual block.

2.2 Feature Reuse Opportunity

Our strategy for accelerating streaming point-cloud perception is to reuse intermediate features from previously observed spatial regions rather than repeatedly encoding them. Effective feature reuse depends on two questions: (1) how many tokens overlap across consecutive frames, and (2) whether the features of these overlapping tokens can be reliably reused. We examine these questions through temporal overlap and semantic consistency of intermediate features across frames.

Temporal overlap of intermediate tokens. Feature reuse is beneficial only when many tokens correspond to spatial regions already observed in previous frames. We therefore quantify their overlap across consecutive frames at each encoder stage. Because progressive downsampling changes the spatial granularity represented by each token, the degree of cross-frame overlap can vary substantially across stages.

To quantify how often tokens from different frames correspond to the same physical regions, we transform their coordinates into a shared global coordinate system so that they can be compared in a common spatial reference frame (details; Sec. 3.2). Figure 2(b) shows that temporal overlap increases substantially with network depth: from 4% to 55% on nuScenes, 14% to 75% on Waymo, and 57% to 88% on ScanNet. Progressive downsampling makes deeper tokens represent larger spatial regions, increasing the likelihood that the same regions are observed across consecutive frames.

Semantic consistency of overlapping features. Spatial overlap alone does not guarantee reliable feature reuse; features from overlapping regions must also preserve consistent semantic information

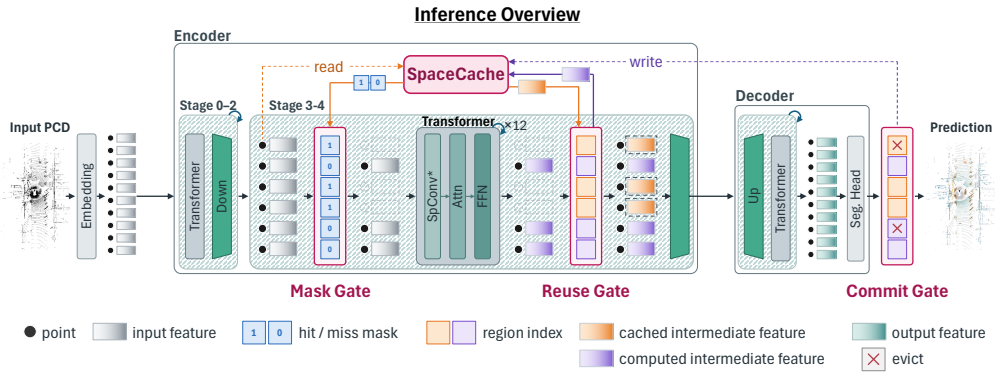


Figure 3: **Overview of SpaceCache.** Inference pipeline with the Mask, Reuse, and Commit Gates. Cache-hit tokens bypass the selected transformer blocks, while cache-miss tokens are computed through the blocks. The Reuse Gate restores the full token layout, and the Commit Gate updates the cache after prediction.

across frames. In other words, a feature computed for a region in a previous frame should remain representative of the same semantic category when reused in the current frame.

To quantify this consistency, we define "Reuse Agreement". For each semantic category c, we first construct a mean feature μ_c by averaging the features f_i associated with that category. We then measure whether the reused feature and its corresponding current-frame feature are associated with the same semantic category (details; Appendix A.2). Figure 2(c) shows that Reuse Agreement increases with encoder depth, from approximately 60% at Stage 0 to 85–88% at Stages 3–4. In particular, the agreement begins to saturate from Stage 3, indicating that cached features at deeper stages largely preserve the semantic information of their corresponding current-frame features.

Design Takeaway. Figures 2(b) and (c) show that deeper encoder stages exhibit both higher temporal overlap and stronger semantic consistency across frames. These properties make deeper intermediate representations increasingly suitable for temporal feature reuse.

3 Method

3.1 Overview of SpaceCache

Figure 3 illustrates the overall inference flow of a point-cloud encoder integrated with SpaceCache. SpaceCache stores reliable intermediate features from the *last* transformer block of selected encoder stages and reuses them when the corresponding spatial regions are observed again (cache-hit) in subsequent frames. The process is implemented through three gates with distinct roles: the Mask Gate, Reuse Gate, and Commit Gate.

The **Mask Gate** queries SpaceCache using the coordinates p_i of current-frame tokens to identify cache hits and misses. Cache-hit tokens (dotted bars in Figure 3) bypass the transformer blocks, while only cache-miss tokens are processed. After the transformer blocks, the **Reuse Gate** combines the newly computed features of cache-miss tokens (purple bars) with the cached features of cache-hit tokens (orange bars) and restores the full-token layout for subsequent layers. Newly computed cache-miss features are also retained as candidates for the cache update. Once the frame prediction is produced, the **Commit Gate** determines which cached entries and candidate features remain valid and updates SpaceCache accordingly.

3.2 Operation Details of SpaceCache

SpaceCache components. Figure 4 illustrates the operations of SpaceCache in detail. Each entry in the cache consists of three components: a *Regional Index* $\mathbf{v} \in \mathbb{Z}^3$ recording a previously observed spatial region, a *Feature* $\mathbf{f} \in \mathbb{R}^n$ storing the intermediate representation computed for that region, and a scalar *Age* $a \in \mathbb{N}$ counting the frames since the entry was inserted. The entries are maintained in arrays sorted by the index, allowing each lookup to be efficiently performed by binary search.

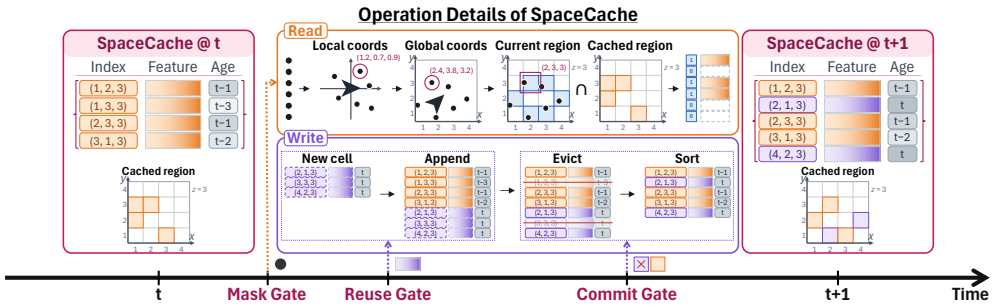


Figure 4: **Operation details of SpaceCache.** Each entry stores a region index, cached feature, and age. **Read:** Current-frame tokens are mapped to global region indices and queried at the Mask Gate to identify cache hits; the corresponding cached features are then retrieved for reuse at the Reuse Gate. **Write:** Newly computed cache-miss features become candidate entries at the Reuse Gate; after prediction, the Commit Gate evicts invalid entries, commits valid candidates, and re-sorts the cache for the next-frame lookup.

Transformation from token coordinate to region index. To reuse features across frames, SpaceCache must determine whether a token corresponds to a previously observed spatial region. However, token coordinates are generally expressed in each frame's *local* coordinate system and hence cannot be directly compared across time. We therefore transform each token coordinate p_i^t into a shared *global* coordinate system using the ego pose T_t and then quantize it into a spatial cell whose size depends on the encoder stage, so that observations of the same physical region across frames are assigned the same region index: $\bar{p}_i^t = T_t p_i^t$ and $v_i^t = \left\lfloor \frac{\bar{p}_i^t}{s_l} \right\rfloor$, where $s_l = 2^l g$ is the cell size at stage l, g is the input voxel size, and 2^l is the corresponding downsampling ratio. Finally, the three-dimensional region index v_i^t is mapped into a scalar key $\phi(v_i^t)$ for efficient cache lookup on the GPU (Appendix B.1).

Step 1: cache read. At the Mask Gate, SpaceCache is queried using the region indices of the current-frame tokens. The lookup returns a cache-hit mask indicating which regions are already present in the cache (denoted as "Current region ∩ Cache region" in Figure 4), together with the cached features of the cache-hit tokens. The Mask Gate uses the hit mask to bypass redundant computation, while the Reuse Gate later retrieves the corresponding cached features (orange bars in "Read" of the figure). This operation is lightweight because it only requires checking whether each region index exists in the cache. Since the cache entries are sorted by region indices, we perform a binary search for each token in parallel on the GPU. Each lookup takes $\mathcal{O}(\log M)$ operations, resulting in $\mathcal{O}(N \log M)$ total work for N input tokens and M cached entries.

Step 2: candidate write. At the Reuse Gate, newly computed features of cache-miss tokens are paired with their region indices and temporarily stored as *candidate* entries (purple bars in "Write" of the figure). These candidates remain separate from the active cache until the frame-level semantic predictions become available for reliability assessment. Meanwhile, the newly computed features are combined with the cached features of cache-hit tokens to restore the original token layout. The candidates are then passed to the Commit Gate for validation and possible insertion into the cache.

Step 3: cache commit and eviction. Intermediate feature reuse does not guarantee that a cached feature remains valid for every subsequent frame. First, cached features may become stale as the scene changes, particularly in regions containing dynamic objects, where the semantic content of the same spatial region can change across frames. Second, a feature may be unreliable when it is first computed, particularly in ambiguous regions such as object boundaries or when reduced context leads to uncertain predictions. Repeatedly reusing such features can propagate errors across subsequent frames. Thus, SpaceCache evaluates on the fly whether newly generated candidate features should be committed to the cache and whether existing cached features should be evicted (Sec. 3.3).

3.3 Cache Management in SpaceCache

Confidence filtering—dynamic commit and eviction. We use decoder prediction confidence as a proxy for feature reliability, filtering out low-confidence features from both new candidates and cached entries. As Figure 5 shows, Reuse Agreement (Sec. 2.2) increases with region confidence for

both candidate and cached entries, indicating that higher-confidence features tend to remain more semantically consistent across frames.

The Commit Gate computes region confidence from the output logits of the current frame. For each point, we take its maximum class probability, and the confidence of a region v is the average of these probabilities over the input points in v:

$$c(v) = \frac{1}{|\Omega_v|} \sum_{j \in \Omega_v} \max_k \sigma(z_j)_k, \quad (1)$$

where Ω_v is the set of input points associated with region v, z_j is the logit vector of point j, and σ is the softmax. With a pre-defined confidence threshold θ, candidate entries with $c(v) < \theta$ are discarded, and cached entries below the same threshold are evicted.

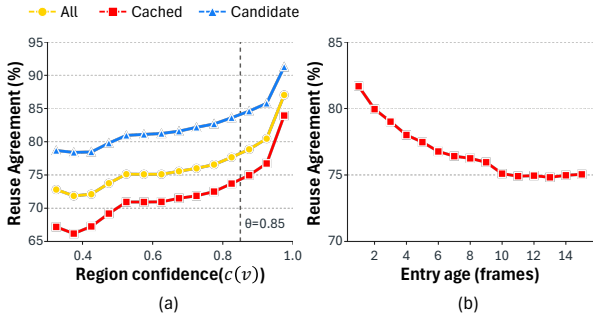


Figure 5: **Reuse Agreement of cache entries** (Stage-3, Waymo with Utonia). (a) By region confidence $c(v)$ at L=8 for all, cached, and candidate entries; the dashed line marks θ=0.85. (b) By entry age with L=∞.

Lifetime–static eviction. Confidence filtering does not explicitly bound how long a cached feature can persist, while Figure 5 shows that reuse agreement decreases with cache age. In particular, older entries are progressively less likely to remain semantically consistent with the corresponding current-frame features. SpaceCache therefore imposes a maximum lifetime L on cache entries. Each newly committed entry is assigned a random expiration age drawn uniformly from $\{1, \ldots, L\}$, preventing all entries from expiring simultaneously. Once expired, the corresponding region is recomputed when observed again. A smaller L refreshes features more frequently at the cost of additional computation. We study this trade-off in Sec. 4.2.

3.4 Computation Reduction and Overhead

Computation reduction. Let N_l be the number of tokens at stage l and h_l the cache-hit rate; after the Mask Gate, the enclosed encoder blocks process only $(1-h_l)N_l$ tokens. The major operations in point-cloud transformer blocks scale linearly with the number of tokens: windowed attention costs $\mathcal{O}(WN_l)$ for a fixed window size W, token-wise projections and MLPs cost $\mathcal{O}(N_l)$, and submanifold sparse convolution costs $\mathcal{O}(KN_l)$ for a fixed kernel size K. Hence, SpaceCache reduces the computation of these operations approximately in proportion to h_l.

Overhead. SpaceCache incurs a one-time per-frame overhead for region-index generation, cache lookup, feature assembly, and cache update. In our setting, all cache operations together take about 5 ms per frame, while the cache occupies approximately 70 MiB with single-precision features. The overhead remains small because index generation uses simple coordinate transformation and quantization, lookup performs parallel binary searches over scalar indices, and feature assembly only restores cached and computed features to their token positions. Since these costs are largely independent of the number of enclosed blocks, they are better amortized as more blocks are bypassed. We provide a detailed breakdown in Sec. 4.3.

Stage selection for SpaceCache. Following the design takeaway from Sec. 2, among the five encoder stages (S0-S4), Stages 3 and 4 exhibit both high temporal overlap and high feature reusability. We therefore primarily apply SpaceCache to these deeper stages. SpaceCache remains applicable to any sequence of consecutive blocks within a stage that preserves token coordinates, and we evaluate different stage selections in Sec. 4.2.

4 Experimental Results

Dataset and Implementation Details. We use the full validation sets of two outdoor LiDAR datasets, nuScenes (Caesar et al., 2020) (6,019 frames, 150 scenes) and Waymo (Sun et al., 2020) (5,976 frames, 202 scenes), whose labeled sweeps arrive at 2 Hz. For the indoor stream, we convert each ScanNet (Dai et al., 2017) RGB-D frame with its camera pose into a point-cloud stream subsampled to 3 Hz, using 10 validation scenes (2,221 frames). We integrate SpaceCache into two pretrained point-cloud networks, Utonia (Zhang et al., 2026) and LitePT-B (Yue et al., 2026). By

Table 1: **Comparison on outdoor streams.** Each method is shown at its fastest setting within a 1-point mIoU drop from the backbone (rings in Figure 6). Latency (ms) and GFLOPs are measured per frame on representative scenes with dense attention. Dashes indicate that no setting satisfies the 1-point drop.

	Waymo				nuScenes			
Method	mIoU (%)↑	Latency↓	GFLOPs↓	Speedup↑	mIoU (%)↑	Latency↓	GFLOPs↓	Speedup↑
Utonia	67.17	721	4438	1.00×	77.29	226	1269	1.00×
+ ToMe*	66.35	691	3794	1.04×	76.31	255	1024	0.89×
+ ALGM*	66.24	633	3751	1.14×	76.36	209	1060	1.08×
+ GitMerge3D†	66.38	706	3864	1.02×	76.32	237	1123	0.95×
+ SpaceCache $_{\text{Stage-3}}$	66.25	563	3177	1.28×	76.47	192	982	1.18×
+ SpaceCache $_{\text{Stage-3,4}}$	66.50	**537**	**2941**	**1.34**×	76.38	**187**	**918**	**1.21**×
LitePT-B	72.82	307	2392	1.00×	80.04	98	683	1.00×
+ ToMe*	72.00	245	1488	1.25×	79.33	101	492	0.97×
+ ALGM*	–	–	–	–	–	–	–	–
+ GitMerge3D†	72.05	287	2020	1.07×	79.05	101	605	0.97×
+ SpaceCache $_{\text{Stage-3}}$	71.86	202	1374	1.52×	79.21	76	468	1.29×
+ SpaceCache $_{\text{Stage-3,4}}$	72.11	**186**	**1226**	**1.65**×	79.14	**75**	**444**	**1.31**×

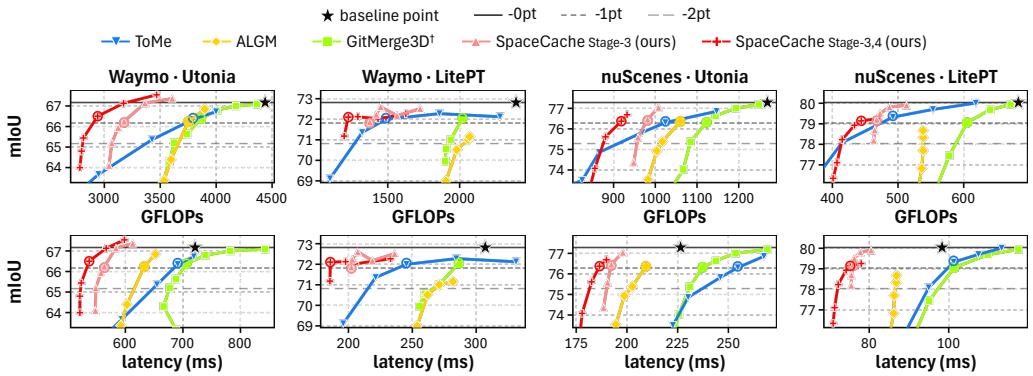


Figure 6: **Accuracy-efficiency trade-off on outdoor streams.** Curves sweep the merge ratio (ToMe, GitMerge3D), similarity threshold (ALGM), or confidence threshold θ (SpaceCache), plotting mIoU against FLOPs (top) and latency (bottom). Solid line and star–backbone; dashed lines–1- and 2-point mIoU drops; Rings–cheapest point of each method within the 1-point drop.

default, SpaceCache is applied to encoder stages 3 and 4 (SpaceCache$_{\text{Stage-3,4}}$); we also evaluate a stage-3-only variant (SpaceCache$_{\text{Stage-3}}$). We set the lifetime L to 8/4 frames for outdoor/indoor streams and the confidence threshold θ to 0.85. Outdoor/indoor latency is measured on an NVIDIA RTX A6000/Jetson AGX Orin. All models use a single forward pass without test-time augmentation.

Baselines. We compare SpaceCache with token-merging methods that aggregate redundant tokens within each frame: ToMe (Bolya et al., 2023) and ALGM (Norouzi et al., 2024), designed for 2D image transformers and applied to point tokens, and GitMerge3D (Tran et al., 2025) without its adaptive patch partition[1]. Implementation details are provided in Appendix B.

4.1 Comparison with Prior Methods

Comparison in outdoor scenario. Table 1 compares SpaceCache with ToMe, ALGM, and GitMerge3D on Waymo and nuScenes with Utonia and LitePT-B, each at its fastest setting within a 1-point mIoU drop. SpaceCache is the fastest in all four settings, reaching up to 1.65× on Waymo with LitePT-B (307→186 ms) by removing 49% of FLOPs.

Figure 6 extends this comparison over the full sweep of each method. Within the 2.0-point drop, SpaceCache achieves a better trade-off than token merging in all four settings, in both FLOPs (top)

[1]The adaptive version reduces each redundant attention patch from 1024 tokens to 32, which drops the accuracy sharply on the outdoor streams, and on the indoor stream its patch partition costs more latency than the merging saves (Appendix B.4).

Table 2: **Comparison on indoor streams.** Each method is shown at its fewest-FLOPs setting within 1-point and 2-point mIoU drops. Kernel time (ms) is averaged over 50 representative frames. The notation follows Table 1.

	mIoU drop $\leq$ 1.0 pt				mIoU drop $\leq$ 2.0 pt			
Method	mIoU (%)↑	Kernel time↓	GFLOPs↓	Speedup↑	mIoU (%)↑	Kernel time↓	GFLOPs↓	Speedup↑
LitePT-B	50.63	43.3	39.1	1.00×	50.63	43.3	39.1	1.00×
+ ToMe*	49.72	41.1	27.4	1.06×	48.80	40.5	26.0	1.07×
+ ALGM*	49.80	39.9	31.4	1.09×	48.75	37.2	29.1	1.16×
+ GitMerge3D†	49.64	47.2	36.6	0.92×	48.65	46.8	36.3	0.93×
+ SpaceCache $_{\text{Stage-3}}$	49.81	**36.8**	**27.3**	**1.18**×	48.68	**35.5**	**24.8**	**1.22**×
+ SpaceCache $_{\text{Stage-3,4}}$	49.68	38.8	30.2	1.12×	48.80	36.2	25.4	1.20×

Table 3: **Ablation study on Waymo with Utonia.** (a) Lifetime control. (b) Stage selection. Gray: default setting; L=1 reproduces the backbone. Hit: cache-hit fraction of each cached stage. ΔDyn./ΔSta.: accuracy change (points) on dynamic/static regions at stage-3 relative to L=1 (83.7/94.8%; Appendix C.3). ΔFLOPs: encoder FLOPs change relative to the backbone.

(a) Cache management (commit and eviction)

	Age only				Age + Confidence			
L	Speedup↑	mIoU↑	Δ Dyn.	Δ Sta.	Speedup↑	mIoU↑	Δ Dyn.	Δ Sta.
1	1.00×	67.18	–	–	1.00×	67.18	–	–
2	1.09×	67.14	−0.5	−0.1	1.10×	67.12	−0.5	0.0
4	1.24×	65.84	−1.9	−0.3	1.21×	67.11	−1.1	0.0
8	1.35×	63.99	−3.0	−0.8	1.25×	66.91	−1.8	0.0
16	1.44×	62.79	−4.1	−1.0	1.33×	66.68	−3.0	0.0
32	1.44×	62.37	−5.3	−0.9	1.37×	66.53	−3.9	−0.1

(b) Stage selection for SpaceCache

Stage	Hit (%)	ΔFLOPs (%)	Speedup↑	mIoU↑
0	20	−0.7	1.00×	67.00
1	32	−2.1	1.01×	65.28
2	42	−5.0	1.03×	65.42
3	48	−34.7	1.21×	66.60
4	52	−6.5	1.03×	67.47
3+4	48 / 51	−41.1	1.25×	66.91

and latency (bottom). Two properties underlie this advantage: more FLOPs removed and better FLOPs-to-latency conversion. We analyze both in Sec. 4.3.

Comparison in indoor scenario. Table 2 reports each method using LitePT-B on ScanNet at its setting with the fewest FLOPs within 1-, 2-point mIoU drops. Since host-side launch overhead dominates the latency of small indoor frames, we report the average GPU kernel time over 50 representative frames. SpaceCache$_{\text{Stage-3}}$ is the fastest with the fewest FLOPs at both drops, reaching 1.22× with 37% fewer GFLOPs within the 2-point drop. The speedup is smaller than outdoors because the denser indoor points are merged more aggressively by downsampling.

4.2 Ablation Study

Cache management. Table 3(a) examines SpaceCache's two control knobs, which trade accuracy for efficiency rather than having a single optimal value: the lifetime L and confidence filtering with threshold θ. With confidence filtering ("Age + Confidence"), increasing L from 1 to 32 raises the speedup to 1.37× but lowers the mIoU by 0.65 points, as older features survive longer in the cache; the drop is more pronounced on dynamic objects (Δ Dyn. of −3.9). Without filtering ("Age only"), the mIoU drops far more steeply (4.81 points at L=32), and even at a similar speedup it trails the filtered setting (65.84% at 1.24× for L=4 vs. 66.91% at 1.25× for L=8).

Stage selection. Table 3(b) shows how performance changes with the stage which SpaceCache is applied to. The speedup grows with the number of enclosed blocks, peaking at stage-3, which holds 12 of the 24 blocks. The accuracy loss shrinks with depth from stage-1 to 4, consistent with the semantic consistency in Figure 2; despite its low consistency, stage-0 loses little, as it reuses the fewest tokens and all later stages refine them. Stage-3, 4 together thus give the largest speedup at a small accuracy drop, which we use as the default.

4.3 Efficiency Analysis

Where the gain comes from. SpaceCache outperforms token merging at similar accuracy (Figure 6) for two reasons, decomposed in Figure 7 on Waymo with Utonia within a 2-point drop. (1) *More FLOPs removed.* Cache-hit tokens skip every block of a stage, so SpaceCache shrinks the sparse convolution, attention, and FFN together (Figure 7(a)), removing 37% of the total FLOPs. GitMerge3D instead merges tokens only within attention, and its per-block scoring leaves a net reduction of only 18%. (2) *Better FLOPs-to-latency conversion.* SpaceCache runs its cache opera-

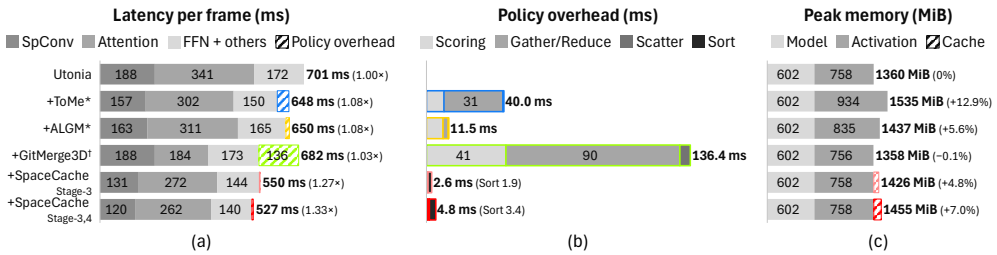


Figure 7: **Latency and memory breakdown** on the representative scenes of Waymo with Utonia, each method within a 2.0-point mIoU drop. (a) Latency per frame, split into the backbone layers and the policy overhead. (b) Policy overhead per frame, split into its components. (c) Peak memory with flash-attention, split into the model weights, the activation memory, and the cache.

tions once per stage, costing only 4.8 ms, mostly for sorting at the Commit Gate. Merging repeats scoring at every block and accumulates merged features through memory-bound gather/reduce, so GitMerge3D adds 136 ms of overhead (Figure 7(b)). Even ToMe, which removes more FLOPs than SpaceCache$_{\text{Stage-3}}$ on Waymo with LitePT-B (45% vs. 43%), reaches only 1.39× against 1.52×.

Memory breakdown and overhead. The memory overhead of SpaceCache is also small. As shown in Figure 7(c), SpaceCache raises the peak memory by only 95 MiB (7.0%): 69 MiB for the cache itself and 26 MiB of activations for holding cache-miss features until the Commit Gate.

5 Related Work

Efficient point-cloud backbones. Point-cloud backbones have moved from sparse convolution (Graham et al., 2018; Choy et al., 2019) to point transformers (Zhao et al., 2021; Wu et al., 2022), and many designs make them faster by sampling points randomly (Hu et al., 2020), grouping points into superpoints (Robert et al., 2023), or simplifying attention (Wu et al., 2024; Yue et al., 2026). Such designs, however, need training, which is costly now that backbones are pretrained at scale (Wu et al., 2025; Zhang et al., 2026). We instead speed up existing pretrained backbones on point-cloud streams without training.

Token reduction in vision. A common way to speed up transformer networks in vision is to process fewer tokens. Token merging combines similar tokens (Bolya et al., 2023; Liang et al., 2022a; Norouzi et al., 2024; Tran et al., 2025), pruning removes unimportant ones (Liang et al., 2022b; Chen et al., 2023; Zhao et al., 2023; Baiocchi et al., 2025; Li et al., 2025), and halting stops tokens at intermediate layers in 2D (Yin et al., 2022; Tang et al., 2023) and 3D (Ye et al., 2023). Some point networks instead group points into fewer tokens (Xu et al., 2021; Wang et al., 2025). These methods require training by design, except for merging. Merging, however, often searches for similar tokens again at every block and accumulates several tokens into one through slow memory accesses, which limits its speed-up (Lu et al., 2025). SpaceCache instead selects tokens cheaply and only once per stage, and copies each cached feature one-to-one without accumulating tokens.

Temporal reuse in point-cloud streams. While many methods use past frames for accuracy (Liu et al., 2019; Cao et al., 2020; Li et al., 2023; He et al., 2023; Dang et al., 2025), few use them to save computation. Of these, TempNet (Zhou et al., 2021) propagates key-frame features with trained modules, and SSCATeR (Dow et al., 2026) recomputes convolutions only where points changed. Closest to our work, EVE (Peng et al., 2021) prunes input points near the previous frame and trains a module to recover lost context. SpaceCache instead keeps all input points, reuses only deep-stage features, and needs no training.

6 Conclusion

We presented SpaceCache, a training-free intermediate-feature-reuse method for efficient point-cloud stream processing. By exploiting the high temporal overlap and feature reusability of deeper encoder stages, SpaceCache skips redundant computation while preserving dense token representations. Across outdoor and indoor semantic-segmentation benchmarks, SpaceCache consistently improves the accuracy-efficiency trade-off over existing token-reduction methods, with low runtime and memory overhead.

AI USE STATEMENT

In this work, we used generative AI tools for implementing the methods designed by the authors in the codebase and for assisting with translation. We have not used generative AI tools to develop theoretical models or conceptual frameworks, formulate mathematical claims or proofs, propose or refine hypotheses, design or provide feedback on the research methodology or experiments, or interpret results, and the remaining required-disclosure tasks (generating synthetic data, dataset cleaning, and qualitative data analysis) are not applicable to this work. Additionally, we used generative AI tools for creating and editing software code, including the plotting code for the figures, drafting and editing parts of the manuscript, identifying relevant literature, and formatting references. We have reviewed all AI-assisted work. All AI-generated code was verified and tested for correctness by the authors, and all AI-suggested references were checked against the original publications. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

ETHICS STATEMENT

This work uses only the public Waymo Open dataset, nuScenes, and ScanNet under their respective licenses, and involves no human subjects or newly collected data. Since SpaceCache trades a small accuracy loss for lower latency in perception systems that may be safety-critical, such as autonomous driving, its accuracy should be validated in the target environment before deployment.

REPRODUCIBILITY STATEMENT

The code is provided as supplementary material. It builds on the Pointcept codebase (Pointcept Contributors, 2023) and contains SpaceCache, our implementations of the baselines, the evaluation configurations, and one script per table and figure that reproduces the reported numbers from the public datasets and checkpoints. SpaceCache requires no training, and the backbones are trained with the included Pointcept configurations and shared unchanged by all methods. Section 4 gives the datasets, backbones, baselines, and default hyperparameters of SpaceCache. Appendix A defines the token regions and the Reuse Agreement used in the motivation study. Appendix B describes the implementation of SpaceCache and the three baselines, and Table 4 lists the swept values of every method. Appendix C describes how the ScanNet video stream is built from the raw RGB-D videos, the representative scenes and the measurement protocol for latency, FLOPs, and memory, and the metrics of the ablation study. Appendix D details the latency breakdown of Sec. 4.3.

REFERENCES

Alessandro Baiocchi, Indro Spinelli, Alessandro Nicolosi, and Simone Scardapane. Adaptive token selection for scalable point cloud transformers. *Neural Networks*, 188:107477, 2025.

Jens Behley, Martin Garbade, Andres Milioto, Jan Quenzel, Sven Behnke, Cyrill Stachniss, and Jürgen Gall. SemanticKITTI: A dataset for semantic scene understanding of LiDAR sequences. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, 2019.

Daniel Bolya, Cheng-Yang Fu, Xiaoliang Dai, Peizhao Zhang, Christoph Feichtenhofer, and Judy Hoffman. Token merging: Your ViT but faster. In *International Conference on Learning Representations (ICLR)*, 2023.

Holger Caesar, Varun Bankiti, Alex H. Lang, Sourabh Vora, Venice Erin Liong, Qiang Xu, Anush Krishnan, Yu Pan, Giancarlo Baldan, and Oscar Beijbom. nuScenes: A multimodal dataset for autonomous driving. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020.

Hanwen Cao, Yongyi Lu, Cewu Lu, Bo Pang, Gongshen Liu, and Alan Yuille. ASAP-Net: Attention and structure aware point cloud sequence segmentation. In *Proceedings of the British Machine Vision Conference (BMVC)*, 2020.

Xuanyao Chen, Zhijian Liu, Haotian Tang, Li Yi, Hang Zhao, and Song Han. SparseViT: Revisiting activation sparsity for efficient high-resolution vision transformer. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2023.

Christopher Choy, JunYoung Gwak, and Silvio Savarese. 4D spatio-temporal ConvNets: Minkowski convolutional neural networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019.

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/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

Chenxu Dang, Zaipeng Duan, Pei An, Xinmin Zhang, Xuzhong Hu, and Jie Ma. FASTer: Focal token acquiring-and-scaling transformer for long-term 3D object detection. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2025.

Tri Dao, Daniel Y. Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. FlashAttention: Fast and memory-efficient exact attention with IO-awareness. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2022.

Alexander Dow, Manduhu Manduhu, Matheus Santos, Ben Bartlett, Gerard Dooly, and James Riordan. SSCATeR: Sparse scatter-based convolution algorithm with temporal data recycling for real-time 3-D object detection in LiDAR point clouds. *IEEE Sensors Journal*, 26(6):7853–7874, 2026.

Benjamin Graham, Martin Engelcke, and Laurens van der Maaten. 3D semantic segmentation with submanifold sparse convolutional networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

Tong He, Pei Sun, Zhaoqi Leng, Chenxi Liu, Dragomir Anguelov, and Mingxing Tan. LEF: Late-to-early temporal fusion for LiDAR 3D object detection. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2023.

Qingyong Hu, Bo Yang, Linhai Xie, Stefano Rosa, Yulan Guo, Zhihua Wang, Niki Trigoni, and Andrew Markham. RandLA-Net: Efficient semantic segmentation of large-scale point clouds. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020.

Alex H. Lang, Sourabh Vora, Holger Caesar, Lubing Zhou, Jiong Yang, and Oscar Beijbom. PointPillars: Fast encoders for object detection from point clouds. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019.

Enxu Li, Sergio Casas, and Raquel Urtasun. MemorySeg: Online LiDAR semantic segmentation with a latent memory. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, 2023.

Jianhao Li, Tianyu Sun, Xueqian Zhang, Zhongdao Wang, Bailan Feng, and Ke Xu. Efficient 3D perception on multi-sweep point cloud with Gumbel spatial pruning. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 2025.

Weicong Liang, Yuhui Yuan, Henghui Ding, Xiao Luo, Weihong Lin, Ding Jia, Zheng Zhang, Chao Zhang, and Han Hu. Expediting large-scale vision transformer for dense prediction without fine-tuning. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2022a.

Youwei Liang, Chongjian Ge, Zhan Tong, Yibing Song, Jue Wang, and Pengtao Xie. Not all patches are what you need: Expediting vision transformers via token reorganizations. In *International Conference on Learning Representations (ICLR)*, 2022b.

Xingyu Liu, Mengyuan Yan, and Jeannette Bohg. MeteorNet: Deep learning on dynamic 3D point cloud sequences. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, 2019.

Wenbo Lu, Shaoyi Zheng, Yuxuan Xia, and Shengjie Wang. ToMA: Token merge with attention for diffusion models. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2025.

Andres Milioto, Ignacio Vizzo, Jens Behley, and Cyrill Stachniss. RangeNet++: Fast and accurate LiDAR semantic segmentation. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2019.

Narges Norouzi, Svetlana Orlova, Daan de Geus, and Gijs Dubbelman. ALGM: Adaptive local-then-global token merging for efficient semantic segmentation with plain vision transformers. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2024.

Gao Peng, Bo Pang, and Cewu Lu. Efficient 3D video engine using frame redundancy. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)*, 2021.

Pointcept Contributors. Pointcept: A codebase for point cloud perception research. `https://github.com/Pointcept/Pointcept`, 2023.

Damien Robert, Hugo Raguet, and Loic Landrieu. Efficient 3D semantic segmentation with Superpoint Transformer. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, 2023.

Pei Sun, Henrik Kretzschmar, Xerxes Dotiwalla, Aurelien Chouard, Vijaysai Patnaik, Paul Tsui, James Guo, Yin Zhou, Yuning Chai, Benjamin Caine, Vijay Vasudevan, Wei Han, Jiquan Ngiam, Hang Zhao, Aleksei Timofeev, Scott Ettinger, Maxim Krivokon, Amy Gao, Aditya Joshi, Yu Zhang, Jonathon Shlens, Zhifeng Chen, and Dragomir Anguelov. Scalability in perception for autonomous driving: Waymo Open Dataset. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020.

Quan Tang, Bowen Zhang, Jiajun Liu, Fagui Liu, and Yifan Liu. Dynamic token pruning in plain vision transformers for semantic segmentation. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, 2023.

Philippe Tillet, H. T. Kung, and David Cox. Triton: An intermediate language and compiler for tiled neural network computations. In *Proceedings of the 3rd ACM SIGPLAN International Workshop on Machine Learning and Programming Languages (MAPL)*, 2019.

Hoai-Chau Tran, Duy M. H. Nguyen, Duy M. Nguyen, TrungTin Nguyen, Ngan Le, Pengtao Xie, Daniel Sonntag, James Zou, Binh T. Nguyen, and Mathias Niepert. Accelerating transformers with spectrum-preserving token merging. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2024.

Tuan Anh Tran, Duy M. H. Nguyen, Hoai-Chau Tran, Michael Barz, Khoa D. Doan, Roger Wattenhofer, Ngo Anh Vien, Mathias Niepert, Daniel Sonntag, and Paul Swoboda. How many tokens do 3D point cloud transformer architectures really need? In *Advances in Neural Information Processing Systems (NeurIPS)*, 2025.

Xuying Wang, Yunsheng Zhang, Run Shao, Siyang Chen, Haifeng Li, and Xin Chen. SAT-Former: An efficient 3-D transformer with semantic aggregated point tokenizer for point cloud semantic segmentation in urban scenes. *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, 18:28326–28342, 2025.

Xiaoyang Wu, Yixing Lao, Li Jiang, Xihui Liu, and Hengshuang Zhao. Point Transformer V2: Grouped vector attention and partition-based pooling. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2022.

Xiaoyang Wu, Li Jiang, Peng-Shuai Wang, Zhijian Liu, Xihui Liu, Yu Qiao, Wanli Ouyang, Tong He, and Hengshuang Zhao. Point Transformer V3: Simpler, faster, stronger. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2024.

Xiaoyang Wu, Daniel DeTone, Duncan Frost, Tianwei Shen, Chris Xie, Nan Yang, Jakob Engel, Richard Newcombe, Hengshuang Zhao, and Julian Straub. Sonata: Self-supervised learning of reliable point representations. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2025.

Chenfeng Xu, Bohan Zhai, Bichen Wu, Tian Li, Wei Zhan, Peter Vajda, Kurt Keutzer, and Masayoshi Tomizuka. You Only Group Once: Efficient point-cloud processing with token representation and relation inference module. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2021.

Mao Ye, Gregory P. Meyer, Yuning Chai, and Qiang Liu. Efficient transformer-based 3D object detection with dynamic token halting. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, 2023.

Hongxu Yin, Arash Vahdat, Jose M. Alvarez, Arun Mallya, Jan Kautz, and Pavlo Molchanov. A-ViT: Adaptive tokens for efficient vision transformer. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2022.

Yuanwen Yue, Damien Robert, Jianyuan Wang, Sunghwan Hong, Jan Dirk Wegner, Christian Rupprecht, and Konrad Schindler. LitePT: Lighter yet stronger point transformer. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2026.

Yujia Zhang, Xiaoyang Wu, Yunhan Yang, Xianzhe Fan, Han Li, Yuechen Zhang, Zehao Huang, Naiyan Wang, and Hengshuang Zhao. Utonia: Toward one encoder for all point clouds. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2026.

Hengshuang Zhao, Li Jiang, Jiaya Jia, Philip Torr, and Vladlen Koltun. Point Transformer. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, 2021.

Tianchen Zhao, Xuefei Ning, Ke Hong, Zhongyuan Qiu, Pu Lu, Yali Zhao, Linfeng Zhang, Lipu Zhou, Guohao Dai, Huazhong Yang, and Yu Wang. Ada3D: Exploiting the spatial redundancy with adaptive inference for efficient 3D object detection. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, 2023.

Yunsong Zhou, Hongzi Zhu, Chunqin Li, Tiankai Cui, Shan Chang, and Minyi Guo. TempNet: Online semantic segmentation on large-scale point cloud series. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, 2021.

A Token Regions and Reuse Agreement

A.1 Token Regions

Pooling chain. Each downsampling layer merges the tokens that fall into the same voxel of the next stage into one token and records which token each of them went to, $\pi_l(i)$ for a token i of stage $l-1$. We call the chain of these records the *pooling chain*. Following it from the input points, the tokens of stage 0, every input point j reaches exactly one token of stage l, so each token v, which represents one spatial region, covers the input points

$$\Omega_v = \{\, j : \pi_l(\pi_{l-1}(\cdots \pi_1(j))) = v \,\}. \tag{2}$$

Token labels. The label of a token is the majority ground-truth label of its input points, $y_v = \arg\max_c |\{j \in \Omega_v : y_j = c\}|$, ignoring the ignore label, and a token is *pure* if all of them carry the same label. The following sections use the same pooling chain and labels.

Token accuracy. Our accuracy measures score a group of tokens by the input points they cover through the pooling chain. For a group G_t of tokens in a frame t, the token accuracy is

$$\mathrm{Acc}(G_t) = \frac{\sum_{v \in G_t} \sum_{j \in \Omega_v} \mathbf{1}[\hat{y}_j = y_j]}{\sum_{v \in G_t} |\Omega_v|}, \tag{3}$$

where $\hat{y}_j$ is the predicted class, the argmax of the logits, and y_j is the ground-truth label of the voxel of point j; points with the ignore label are excluded. It is the recall of the points covered by the group within the frame, with each token weighted by its covered-point count. We then average $\mathrm{Acc}(G_t)$ over all frames, weighted by the covered-point count of each frame. Appendix C.3 uses this measure with different groups G_t.

A.2 Reuse Agreement

Reuse Agreement (Figure 2(c)) measures whether the feature cached for a region in frame $t-1$ points to the same classes as the feature computed for that region in frame t.

Class prototypes. At stage l, the prototype of class c is the mean current-frame feature over F_c^l, the pure tokens of class c:

$$\mu_c^l = \frac{1}{|F_c^l|} \sum_{f \in F_c^l} f. \tag{4}$$

Only classes with at least 20 such tokens in the frame get a prototype.

Agreement. Each feature f is described by its class profile, the softmax of its cosine similarities to the prototypes,

$$q(f) = \mathrm{softmax}\big([\cos(f, \mu_1^l), \ldots, \cos(f, \mu_C^l)] \,/\, T_l\big). \tag{5}$$

The agreement of a reused token is the overlap of the profiles of its current feature f^t and its cached feature f^{t-1}:

$$\mathrm{RA} = \sum_c \min\left(q_c(f^t),\, q_c(f^{t-1})\right) \in [0, 1]. \tag{6}$$

We average RA over all reused tokens, *i.e.*, the tokens whose region was observed in frame $t-1$, of 60 consecutive frame pairs per dataset, and plot the mean over the three datasets for each backbone. T_l is calibrated per stage so that the class profiles have the same mean normalized entropy of 0.5 at every stage, for a fair comparison between stages.

B Implementation Details

This section describes how SpaceCache and the three baselines are implemented. All methods are implemented in Pointcept (Pointcept Contributors, 2023)[2] and use the same pretrained weights without training, and each baseline is swept over its main parameter (Figure 6; Table 4 lists the swept values). On LitePT-B (Yue et al., 2026), whose attention layers are only in the last two encoder stages, ToMe and GitMerge3D merge in those blocks.

B.1 SpaceCache

Settings. The region of each stage is its own voxel, $s_l = 2^l g$, with $g = 5\,\text{cm}$ on Waymo (Sun et al., 2020) and nuScenes (Caesar et al., 2020) and 2 cm on ScanNet (Dai et al., 2017). The cache is emptied at the start of every sequence. The lookup, the feature assembly, and the cache update run as fused Triton (Tillet et al., 2019) kernels on the RTX A6000 and in PyTorch on the Jetson AGX Orin.

Region key. ϕ shifts the three indices of a region $v = (v_x, v_y, v_z)$ to non-negative values and packs them into separate bit fields of one 64-bit integer, 24 bits for v_x and v_y and 15 bits for v_z:

$$\phi(v) = \big((v_x + 2^{23}) \ll 39\big) \,|\, \big((v_y + 2^{23}) \ll 15\big) \,|\, (v_z + 2^{14}), \tag{7}$$

where $\ll$ is the left bit shift and $|$ the bitwise OR. It costs three additions, two shifts, and two ORs per token, and it is one-to-one for $|v_x|, |v_y| < 2^{23}$ and $|v_z| < 2^{14}$, *i.e.*, within $\pm 2^{23} s_l$ horizontally and $\pm 2^{14} s_l$ vertically of the global origin. The range doubles with each stage:

Dataset	Stage	Cell s_l	Horizontal	Vertical
Waymo, nuScenes	3	0.4 m	±3,355 km	±6.5 km
ScanNet	3	0.16 m	±1,342 km	±2.6 km
Waymo (placement ablation)	0	5 cm	±419 km	±819 m

All are far larger than any sequence in the benchmarks.

Lifetime. The Age of an entry is implemented as a due frame rather than a counter. Each region index v has a fixed phase, the linear hash $p_1 v_x + p_2 v_y + p_3 v_z$ with large primes p_i, and an entry committed at frame t is due at the next frame congruent to that phase modulo L: $d_v = t + 1 + (\text{phase}(v) - (t{+}1)) \bmod L$. The lookup counts an entry as a cache hit only while $t < d_v$. Since the due frame is modular in the region index instead of counted from the insertion frame, every region is refreshed at a fixed L-frame cadence and roughly one L-th of the resident entries expires per frame, which keeps the cache-hit ratio stable across frames.

Regions with ignore labels. On Waymo and nuScenes, the points close to the ego vehicle carry the ignore label and receive no training loss, so the features of the regions that contain them are unreliable. The Commit Gate therefore does not insert a newly computed region that contains such a point. This reflects a fixed property of the datasets: the same regions could be removed in advance without any label, *e.g.*, as a fixed area around the ego vehicle. The ignored points never count toward the mIoU, and the rule is not used on ScanNet.

B.2 ToMe

ToMe (Bolya et al., 2023)[3] merges tokens by bipartite matching between the attention and the MLP of every attention block. Within each serialized attention patch, the tokens are split alternately into two sets, each token of the first set is matched to its most similar token of the second set by the cosine similarity of their attention keys, and the best-matched pairs are averaged into one token. The sweep parameter r is the merging ratio, the fraction of the tokens of each patch merged in a block. A token merged at stage l stays merged until the decoder unpooling from stage l to $l{-}1$, which copies its feature back to its original tokens. Stage 0 has no unpooling below it, so its merges are unmerged at the end of the decoder, before the segmentation head.

[2] https://github.com/Pointcept/Pointcept

[3] https://github.com/facebookresearch/ToMe

B.3 ALGM

ALGM (Norouzi et al., 2024)[4] merges twice. Local merging (CLAP), after the attention of the first block of stage 0, merges each $4\times4\times4$-voxel window into one token if the mean pairwise cosine similarity of its tokens exceeds τ. The windows are offset by two voxels from the downsampling grid: an aligned window holds exactly the tokens that the next two downsamplings merge into one, so the downsampling would absorb its merge. Global merging (GBM), after the attention of the first block of stage 3, splits all tokens of the stage alternately into two sets and matches them by bipartite matching. Following the official implementation, it keeps a pair only if its similarity exceeds the mean of the best-match similarities plus their standard deviation divided by the block index, so the number of merges adapts to each frame. Both merges stay until the decoder returns to their stage and copies the merged features back: GBM at the first decoder block of stage 3, and CLAP at the first decoder block of stage 0 (on LitePT-B, which has no decoder blocks, at the corresponding unpooling).

B.4 GitMerge3D

GitMerge3D (Tran et al., 2025)[5] merges inside the attention of every attention block. Each serialized attention patch is divided into bins of $\lceil 1/(1-r) \rceil$ tokens with one random destination token each. The similarity score is the cosine similarity of the value vectors, computed per attention head, and the fraction r of the tokens of the patch with the highest score to a destination is averaged into its most similar destination. The queries, keys, and values are merged this way, the attention runs on the merged tokens, and its output is unmerged right away by copying each merged output back to its original tokens. As in the official implementation, the merge accumulates the tokens through atomic memory updates. We report GitMerge3D without the adaptive patch partition († in the tables). The adaptive version splits the patches by a global energy score into informative patches, merged at ratio r, and redundant patches, reduced to 32 tokens each. On the outdoor streams, reducing a 1024-token patch to 32 tokens is too aggressive and the accuracy drops sharply. On the indoor stream, the adaptive version reduces the FLOPs, but computing the energy and splitting the patches into two buckets costs more latency than the merging saves.

B.5 Swept Settings

Each method is swept over one parameter: the merge ratio r for ToMe and GitMerge3D, the similarity threshold τ of the local merge for ALGM, and the confidence threshold θ for SpaceCache. SpaceCache runs at $L{=}8$ on the outdoor streams and $L{=}4$ on the indoor stream. Table 4 lists the swept values.

Table 4: **Swept parameter values.** The outdoor columns list every point of the Figure 6 curves; the ScanNet column lists the two reported operating points of Table 2. *off* disables the confidence gate. **Bold** marks the point picked within a 1.0-point mIoU drop.

		Waymo		nuScenes		ScanNet
Method	Param.	Utonia	LitePT-B	Utonia	LitePT-B	LitePT-B
ToMe	r	0.01, **0.015**, 0.025, 0.04, 0.045	0.01, 0.05, **0.095**, 0.12, 0.16, 0.2	0.01, **0.02**, 0.025, 0.035, 0.045, 0.05	0.02, 0.04, **0.06**, 0.09, 0.12	**0.11**, 0.13
ALGM	τ	0.96, 0.97, **0.995**, 1	0.6, 0.7, 0.75, 0.85, 0.95, 0.97	0.7, 0.95, 0.97, **0.995**	0.9, 0.925, 0.95, 0.98, 1	**0.9**, 0.8125
GitMerge3D	r	0.1, 0.2, 0.3, **0.4**, 0.5, 0.6, 0.65, 0.72	**0.5**, 0.645, 0.7, 0.72	0.1, 0.2, 0.3, **0.35**, 0.45, 0.5, 0.6	0.1, 0.2, **0.33**, 0.45, 0.6	**0.55**, 0.645
SpaceCache Stage-3	θ	*off*, 0.55, 0.7, **0.8**, 0.95, 0.99	***off***, 0.7, 0.8, 0.95, 0.99	0.8, 0.9, **0.97**, 0.99	*off*, 0.5, **0.65**, 0.8, 0.95, 0.99	**0.85**, *off*
SpaceCache Stage-3,4	θ	*off*, 0.5, 0.55, **0.8**, 0.95, 0.99	*off*, **0.6**, 0.8, 0.95, 0.99	0.7, 0.85, 0.95, **0.985**, 0.99	0.5, 0.7, 0.825, 0.95, **0.97**, 0.99	**0.99**, 0.8

[4] https://github.com/tue-mps/algm-segmenter

[5] https://github.com/anhtuanhsgs/GitMerge3D

C Dataset and Experimental Setup

C.1 ScanNet Video Stream

Frames. We keep every tenth frame (3 Hz) of the 30 Hz RGB-D videos of 10 validation scenes, 2,221 frames in total, and drop the few frames without a valid camera pose. Each depth pixel closer than 8 m is back-projected with the depth intrinsics and takes the color of the nearest color pixel. The scenes are scene0025_00, scene0050_00, scene0169_00, scene0217_00, scene0426_00, scene0488_00, scene0565_00, scene0568_00, scene0580_00, and scene0583_00.

Labels. Each point takes the label of its pixel in the 2D label images of ScanNet (Dai et al., 2017), mapped to the 20 benchmark classes as for the fused scenes. All other labels are ignored.

Coordinates. The backbone is pretrained on the fused ScanNet scenes, so we move every frame to where its points lie in the fused scene: the camera pose maps the frame into the scene frame, and the same offset as for the fused scene is subtracted. The voxel grid is also built in this frame, so all frames of a sequence share one grid.

Surface normals. LitePT-B is pretrained on the smooth normals of the fused-scene meshes, so each point of the stream takes the normal of the nearest mesh vertex.

C.2 Measurement

All methods run with batch size 1 and dense attention instead of flash-attention (Dao et al., 2022). SpaceCache itself is compatible with flash-attention, while the merging baselines merge tokens inside attention and need the attention scores computed over exactly the merged tokens; the dense path provides this on our serialized layout. The attention is computed in bfloat16 for Utonia (Zhang et al., 2026) and float16 for LitePT-B, as in each backbone, and all other layers in float32.

Scenes. Accuracy uses the full validation sets. Latency, FLOPs, and memory use representative scenes, whole sequences whose per-frame distribution of voxel counts matches the full set: 10 segments (299 frames) of Waymo, 8 scenes (321 frames) of nuScenes, and 2 scenes (395 frames) of ScanNet.

Latency. For the outdoor scenes we measure on an RTX A6000 and report the mean frame latency. For the indoor scenes we measure on a Jetson AGX Orin and report the mean GPU kernel time per frame over 50 frames, every eighth frame of the scenes, each replayed under a piecewise CUDA graph. The scenes are streamed in full, so SpaceCache reaches these frames with its true cache state.

FLOPs. The FLOPs count the linear layers, the sparse convolutions with their actual input-output pairs, the attention, and the similarity computations of the merging methods, with one multiply-accumulate as two FLOPs. Normalizations, activations, softmax, and gathers are not counted.

C.3 Metrics of the Ablation Study

The ablations in Table 3 run on Waymo with Utonia. Table 3(a) applies $\text{SpaceCache}_{\text{Stage-3,4}}$ and varies the lifetime L, without confidence filtering (Age only) or with $\theta{=}0.85$ (Age + Confidence). Table 3(b) applies SpaceCache to the listed stages with $L{=}8$ and $\theta{=}0.85$. The mIoU and the region accuracies use the full validation set, and the speedup and the FLOPs use the representative scenes with dense attention. Table 5 defines each reported metric.

Dynamic and static tokens. Each token takes its label (Appendix A.1) in every frame, and is compared with its label at its last observation in the same sequence. A token is *dynamic* if the two labels differ and either of them is a movable class (car, truck, bus, other vehicle, motorcyclist, bicyclist, pedestrian, bicycle, or motorcycle), *i.e.*, an object has entered or left its region. A token is *static* if the two labels are the same. First observations and changes between two static classes, which mostly occur at class boundaries, belong to neither group. Both groups depend only on the

Table 5: **Metrics of the ablation study** (Table 3).

Metric	Definition
mIoU	Mean intersection over union over the classes of the dataset.
Speedup	Frame latency of the backbone divided by that of the method.
Hit	Fraction of the tokens of each cached stage found in the cache, averaged over all frames except the first of each sequence, where the cache is empty.
ΔFLOPs	Relative change of the encoder FLOPs from the backbone.
ΔDyn.	Change of the token accuracy (Appendix A.1) on the dynamic tokens of the third stage from $L{=}1$, in points.
ΔStatic	The same on the static tokens of the third stage.

ground truth, so every run in Table 3(a) is scored on the same tokens. $L{=}1$ reproduces the backbone, with 83.7% accuracy on dynamic and 94.8% on static tokens.

D BREAKDOWN DETAILS

Setting. Figure 7 compares every method at its point within a 2.0-point mIoU drop on the representative scenes of Waymo with Utonia. This section explains where each method removes computation (Table 6), where it spends its own overhead, and what it adds to memory (Table 7). FLOPs and latency use dense attention, and memory uses flash-attention.

FLOPs. SpaceCache removes more FLOPs than token merging because its token selection adds almost no FLOPs. It costs only about $9N_l$ multiply-adds once per stage, for rotating each token coordinate by a 3×3 matrix, whereas the similarity matching of GitMerge3D costs $O(N_l W D)$ at every block, with $W{=}1024$ tokens per window and $D{=}432$ channels at the third stage. Table 6 splits the FLOPs of the whole network by component. SpaceCache$_{\text{Stage-3,4}}$ skips 55% and 60% of the tokens in the third and fourth stages and removes 1,625 of the 4,438 GFLOPs (37%), while its token selection adds less than 0.001 GFLOPs. Since the cache-hit tokens skip every block of the stage, the saving reaches every component, including the sparse convolutions and the MLPs. GitMerge3D merges 60% of the attention tokens, but since it restores all tokens right after the attention, only the attention itself shrinks, which removes 937 GFLOPs, and its matching adds 124 GFLOPs back. Its FLOPs thus drop by only 813 GFLOPs (18%). ToMe and ALGM keep the merged tokens across the following blocks, so their saving also reaches every component, but they remove only 23% and 15% of the FLOPs, net of the 13 and 89 GFLOPs that their matching adds.

Overhead operations. Table 7 splits the overhead of each method into four kinds of operations. All of them move token features rather than compute on them, so their cost is set by memory traffic and by how often they run, not by FLOPs.

- *Scoring* decides which tokens to reduce. The merging methods compute token similarities: ToMe between the two halves of each attention patch, ALGM within each local window and then between all tokens of stage 3, and GitMerge3D between each token and the destination tokens of its patch. SpaceCache computes the region index of each token and looks it up in the cache table by binary search.
- *Gather/reduce* builds the reduced token set. The merging methods accumulate each group of source tokens into its destination with atomic additions, as in their official implementations (Bolya et al., 2023; Norouzi et al., 2024; Tran et al., 2025). Many source tokens add into the same destination at once, and the GPU serializes these conflicting writes. SpaceCache instead gathers the cache-miss tokens into a contiguous subset, reading each token once without any accumulation.
- *Scatter* restores the full token layout. The merging methods copy each merged feature back to its source tokens. SpaceCache writes the newly computed features of the cache-miss tokens and the cached features of the cache-hit tokens to their own positions.
- *Sort* maintains the cache and is used only by SpaceCache. The Commit Gate merges the new entries into the table, which is kept sorted by region key for the binary search, and evicts the expired and low-confidence entries, once per stage and frame.

ToMe and GitMerge3D repeat their scoring, gather/reduce, and scatter at every block, whereas SpaceCache runs each operation once per stage.

Memory. Table 7 splits the peak memory into three parts. The *model* weights take 602 MiB for every method. The *activation* memory is the peak of the intermediate tensors of the forward pass. ToMe and ALGM keep their merge maps and the unmerged tokens until the decoder restores them, which raises the activation memory by 175 MiB and 76 MiB; ToMe merges at every block, so its peak moves from the decoder to encoder Stage 2. GitMerge3D restores the tokens right after each attention and keeps nothing across blocks, so its activation memory stays within 2 MiB of the backbone. SpaceCache adds 18 and 26 MiB of activations, because it holds the newly computed features of the cache-miss tokens until the Commit Gate after the decoder. The *cache* is the persistent table of SpaceCache, one entry per cached region with its 64-bit region key, its feature in single precision (432 channels at the third stage and 576 at the fourth), and its due frame. It takes 47.7 MiB for the third stage and 21.7 MiB for the fourth, 69.4 MiB in total.

Table 6: **FLOPs by component** (GFLOPs per frame, encoder and decoder) in the setting of Figure 7. The first block lists the network layers, which each method shrinks; Overhead (scoring), marked with $+$, is added on top of them and is the FLOPs of the scoring step in Table 7: the similarity matching of the merging methods and the region-index computation of SpaceCache.

Component	Utonia	+ToMe*	+ALGM*	+GitMerge3D†	+SpaceCache Stage-3	+SpaceCache Stage-3,4
Setting	–	r=0.025	τ=0.995	r=0.60	θ=0.55	θ=0.55
CPE (sparse conv. + linear)	1,379.9	964.1	1,067.7	1,379.9	809.7	697.4
QKV projection	481.6	384.9	405.5	481.6	338.3	310.9
Attention ($QK^\top$, AV)	1,116.2	928.5	978.9	178.9	889.7	857.3
Output projection	156.6	128.3	131.3	156.6	108.9	99.7
MLP	1,253.2	969.7	1,034.5	1,253.2	870.9	797.3
Other linear layers	50.2	38.2	43.7	50.2	50.2	50.2
Overhead (scoring)	–	+13.4	+89.3	+124.0	+<0.001	+<0.001
Total	4,437.7	3,427.1	3,750.9	3,624.4	3,067.7	2,812.7
Reduction	–	22.8%	15.5%	18.3%	30.9%	36.6%

Table 7: **Latency and memory breakdown** behind Figure 7 on the representative scenes of Waymo with Utonia, each method at its point within a 2.0-point mIoU drop. Latency (ms per frame) uses dense attention; the frame latency is the sum of the bands and includes the cache update of SpaceCache. For SpaceCache, scoring is the cache lookup, gather/reduce the subset construction, scatter the feature assembly, and sort the cache update at the Commit Gate. Peak memory (MiB) uses flash-attention; the model weights take 602 MiB for every method.

	Utonia	+ToMe*	+ALGM*	+GitMerge3D†	+SpaceCache Stage-3	+SpaceCache Stage-3,4
Setting	–	r=0.025	τ=0.995	r=0.60	θ=0.55	θ=0.55
mIoU (%)	67.17	65.36	66.24	65.21	65.20	65.44
Latency (ms)						
Sparse conv. (CPE)	188.1	156.9	163.2	188.0	131.0	120.2
Attention	340.6	301.9	310.7	184.4	272.0	261.9
FFN + others	171.9	149.7	164.7	173.3	144.2	139.6
Policy overhead	–	40.0	11.5	136.4	2.6	4.8
Scoring	–	9.1	8.6	41.1	0.3	0.5
Gather/reduce	–	30.5	2.7	89.8	0.4	0.8
Scatter	–	0.4	0.2	5.5	0.1	0.2
Sort	–	–	–	–	1.9	3.4
Frame latency	700.6	648.4	650.0	682.0	549.9	526.5
Overhead share	–	6.2%	1.8%	20.0%	0.5%	0.9%
Speedup	1.00×	1.08×	1.08×	1.03×	1.27×	1.33×
Peak memory (MiB)						
Activation	758	934	835	756	776	784
Cache	–	–	–	–	47.7	69.4
Peak	1,360	1,535	1,437	1,358	1,426	1,455
ΔPeak	–	+12.9%	+5.6%	−0.1%	+4.8%	+7.0%