FlashBEV: Fast and Memory-Efficient Exact BEV Transformation with IO-Awareness

Shunsuke Yokokawa[1,2] and Hironori Kasahara[1]

[1] Waseda University, Tokyo, Japan
yokosyun@fuji.waseda.jp, kasahara@waseda.jp
[2] T2, Inc., Tokyo, Japan

Abstract. Bird's-eye-view (BEV) perception is a core component of camera-based 3D understanding in autonomous driving, where view transformation (VT) maps multi-camera image features into a unified BEV representation. Sampling-based view transformation (Sampling-VT) is attractive because it supports dense and continuous BEV aggregation for high-resolution and long-range perception. Its deployment bottleneck, however, is systems-level: standard tensorized implementations of Sampling-VT—which we refer to as Tensorized Sampling-VT—explicitly materialize large height-dependent intermediate tensors, causing memory and latency costs that scale poorly with vertical resolution and the number of cameras.

We revisit Tensorized Sampling-VT from an operator-execution perspective and show that it follows a gather–reduction pattern: each BEV query independently accumulates contributions across cameras and height bins. Unlike splatting-based VT, which requires index sorting and prevents fully thread-local reduction from voxel construction to BEV output, the gather–reduction structure of Sampling-VT enables thread-local accumulation with on-the-fly recomputation, eliminating the need to materialize height- and camera-dependent intermediates.

Based on this insight, we propose **FlashBEV**, a fully fused and IO-aware execution strategy that is mathematically equivalent to Tensorized Sampling-VT (same operator output) while substantially reducing global memory traffic and kernel-launch overhead. Experiments show that FlashBEV achieves more than an order of magnitude lower peak GPU memory and significant inference-latency speedups, with memory usage effectively independent of the number of height bins, reducing the operator's peak memory to $O(BCXY)$ (output only). This unlocks higher BEV range/resolution and vertical discretization within fixed deployment budgets on memory-constrained devices, where tensorized execution would otherwise be infeasible. Our contribution is therefore an execution redesign—same math, different execution—that removes a key scalability barrier for deployment-ready Sampling-VT. Code is available at `https://github.com/yokosyun/FlashBEV`.

Keywords: Bird's-eye-view · View transformation · Memory efficiency · Autonomous driving · GPU optimization

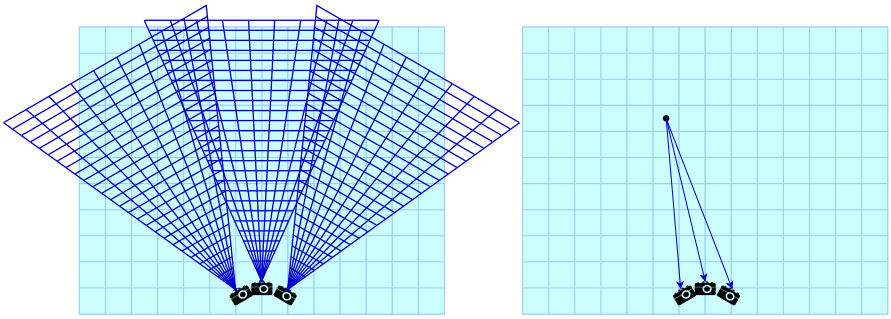

Fig. 1: Conceptual comparison of Splatting-VT (left), which forward-projects (scatters) image features into BEV, and Sampling-VT (right), which backward-queries (gathers) features at each BEV location.

1 Introduction

Bird's-eye-view (BEV) perception is a fundamental representation for camera-based 3D scene understanding in autonomous driving. A central operator in BEV pipelines is *view transformation* (VT), which maps image-space features into a shared BEV coordinate frame. Because VT is executed on every frame, its computational efficiency directly affects end-to-end deployability.

Existing VT designs are broadly divided into *splatting-based VT (Splatting-VT)* and *sampling-based VT (Sampling-VT)*. As shown in Fig. 1, Splatting-VT forward-projects image features into BEV and is often memory-friendly with index-based pooling [11], but it tends to produce geometry-dependent, uneven coverage. Sampling-VT instead performs backward querying from each BEV location into image space, enabling dense and continuous aggregation over the BEV grid. This density makes Sampling-VT attractive for high-resolution and long-range perception.

The challenge is that Sampling-VT is expensive under standard tensorized execution. In this commonly used baseline (*Tensorized Sampling-VT*, i.e., tensorized gather with height-sum reduction), large intermediate tensors are explicitly materialized over BEV cells, cameras, and height bins. Under representative settings (e.g., 50 m BEV range, grid 200×200, 8 height bins, batch size 1), the view-transformation module alone can reach nearly 2 GB peak memory for the forward pass; longer range or higher resolution further increases memory and latency. In practice, this memory-traffic bottleneck, rather than arithmetic throughput, limits deployment. Moreover, while view transformation may not dominate end-to-end latency at standard resolutions, it becomes a primary bottleneck—or triggers out-of-memory failures—as BEV range, BEV resolution, or vertical discretization are scaled.

In this paper, we argue that the bottleneck is not the math of Sampling-VT, but its *execution*. We analyze Tensorized Sampling-VT at operator level

and observe a gather–reduction structure: each BEV query can independently accumulate contributions across cameras and height bins, so the height-expanded tensor $\mathbb{R}^{B\times N\times C\times X\times Y\times Z}$ need not be materialized—sampling and reduction fuse into a thread-local accumulation that directly produces the BEV feature map.

Contributions. Our contribution is an *execution design*, not a new learning algorithm: we preserve the exact Tensorized Sampling-VT operator while changing how it runs on GPU. Specifically, we make the following contributions:

- We identify Tensorized Sampling-VT as a *gather–reduction* computation and expose the source of its memory-traffic inefficiency in tensorized execution.
- We derive a *single fused execution* that performs sampling and accumulation with thread-local reduction and on-the-fly recomputation, eliminating any intermediate tensors.
- We propose **FlashBEV**, an IO-aware implementation of this execution that remains *exactly equivalent* to Tensorized Sampling-VT while reducing memory traffic and kernel overhead.
- We show through experiments that FlashBEV delivers large gains in efficiency—more than an order of magnitude lower peak memory and substantial latency speedups—with memory usage effectively independent of the number of height bins.

To the best of our knowledge, FlashBEV is the first method that executes dense Sampling-VT without materializing camera- and height-dependent tensors and without relying on lookup tables; by removing this fundamental memory bottleneck, FlashBEV opens a new design space for scalable BEV perception and future algorithmic exploration. FlashBEV is orthogonal to model design and can replace dense Sampling-VT in existing pipelines (e.g., SimpleBEV); extension to multi-resolution gathering is future work.

2 Related Work

2.1 Splatting vs. Sampling

We distinguish *Splatting-VT* (scatter) [5,6,9–12,16,20–22,24,25,30] from *Sampling-VT* (gather) [2, 4, 8, 13, 15, 17, 26–29]. Tensorized implementations of Sampling-VT materialize height-dependent intermediate tensors; to our knowledge, no prior work removes this tensor while preserving exact Tensorized Sampling-VT (gather) semantics.

2.2 Sparse Sampling

Several works reduce the computational cost of Sampling-VT by evaluating only a subset of spatial or vertical samples instead of performing dense aggregation. These strategies include restricting computation to selected BEV regions, coarse-to-fine or adaptive sampling schemes, and learned sparse sampling mechanisms such as deformable attention that predict sampling offsets [31].

Sparse BEV Sampling. A number of works reduce the cost of dense BEV querying by activating only a subset of BEV cells or refining selected regions. PointBeV [2] adopts a coarse-to-fine strategy and activates only a subset of BEV cells, reducing computation and memory compared to dense grids. FB-BEV [18] refines only foreground BEV regions by combining forward and backward view transformations, reducing redundant computation over background areas. SparseBEV [19] avoids dense BEV construction entirely via a fully sparse, query-based detector with adaptive spatio-temporal sampling from multi-camera features.

Sparse Height Sampling. Another line of work reduces the cost of Sampling-VT by decreasing the number of height samples (or 3D points) evaluated per BEV location. Instead of uniformly discretizing the vertical dimension, these approaches adaptively allocate height samples according to geometric priors or object distributions. HV-BEV [26] further introduces height-aware reference point sampling, decoupling horizontal feature aggregation and vertical sampling so that the number of sampled heights better matches object height distributions in the scene. These approaches reduce computation by sparsifying vertical sampling, but they still aggregate over multiple height samples or learned offsets per BEV location (e.g., deformable attention over offsets and multi-scale features), and typically maintain intermediate tensors that incur additional memory cost under tensorized execution when full-resolution aggregation is required.

Both lines of work trade full dense BEV coverage (or dense vertical sampling) for sparsification and do not address the *implementation-level* inefficiency of Tensorized Sampling-VT when full-resolution aggregation is required.

2.3 Lookup-Table Based Sampling-VT

An alternative to dense tensorized execution is to precompute geometric correspondences in a lookup table (LUT) and replace runtime projection with indexed gathering; this reduces compute and memory but typically relies on discretized image coordinates and thus does not preserve exact continuous Sampling-VT semantics. Several works adopt LUT-based projection to this end and significantly accelerate BEV feature construction. LUT-based methods differ along two dimensions: when the LUT is built (online vs. offline) and how geometry is used for aggregation (voxel-based vs. interval-based).

Online vs. offline LUTs. Online LUTs are built at runtime and adapt to camera changes but incur per-frame construction and validity checks. Offline LUTs assume static cameras and avoid per-frame cost but require rebuilding when parameters change.

Voxel-based LUT projection. FastBEV [15] (typically with offline precomputation) precomputes a mapping from each BEV voxel to a corresponding image pixel (single-hit per voxel) and performs feature aggregation via indexed gathering. To simplify the projection, each voxel is associated with a single camera source, so it can only sample from a single camera's features even when a BEV location is viewed by multiple cameras. While this design reduces computation and memory access, it limits multi-view fusion in overlapping or partially

occluded regions. In addition, FastBEV constructs intermediate 3D voxel features before collapsing them to BEV, leading to memory usage that scales with the number of height bins.

Interval-based LUT projection. Other approaches adopt interval-based indexing inspired by BEV pooling operators. For example, DualBEV [14] (often with offline precomputation) aggregates image features into BEV grids using precomputed intervals that specify which pixels contribute to each BEV location. This strategy avoids materializing intermediate 3D voxel volumes and improves memory efficiency compared to the voxel-based LUT approach above. However, it still requires additional indexing structures to store image indices and interval ranges, and uses discretized pixel indices.

Despite their efficiency benefits, LUT-based approaches rely on discretized image coordinates obtained through rounding or quantization of projected pixel locations. This discretization can introduce approximation errors in feature sampling and requires constructing LUT structures that depend on camera parameters. FlashBEV, in contrast, preserves exact continuous Sampling-VT semantics (bilinear interpolation and camera aggregation) with no precomputed LUT and no height-dependent intermediates, using only the same inputs as the tensorized baseline: camera projection matrices $(B, N, 3, 4)$ and image feature maps. It requires no indexing structures that scale with the BEV grid or voxel space.

2.4 Kernel Fusion

Basic kernel fusion is typically limited to element-wise operators (e.g., ReLU, Mul, Add) or short predefined patterns in deployment frameworks such as TensorRT [23] or TVM [3]. FlashAttention [7] shows that reordering computation and selectively recomputing intermediates can *enable* much more aggressive kernel fusion, substantially reducing HBM reads/writes while preserving exact semantics. FlashBEV applies the same idea to Sampling-VT: by reordering voxel construction, voxel projection, and validity handling, it achieves a single fused kernel with thread-local accumulation and no height-dependent intermediate tensors, while remaining exactly equivalent to Tensorized Sampling-VT and using minimal HBM (inputs and output only). FlashBEV thus addresses the gap left by sparsification and LUT-based methods by enabling fully fused execution (Sec. 3) without sparsification or discretization.

3 Method

We first formalize Tensorized Sampling-VT and its design choices, then analyze the intrinsic memory bottleneck under fully tensorized execution. Based on these observations, we propose **FlashBEV**, a reordered execution strategy that eliminates intermediate tensor materialization while preserving exact equivalence to the Tensorized Sampling-VT formulation. Fig. 2 gives an overview of the contrast between the baseline and our approach.

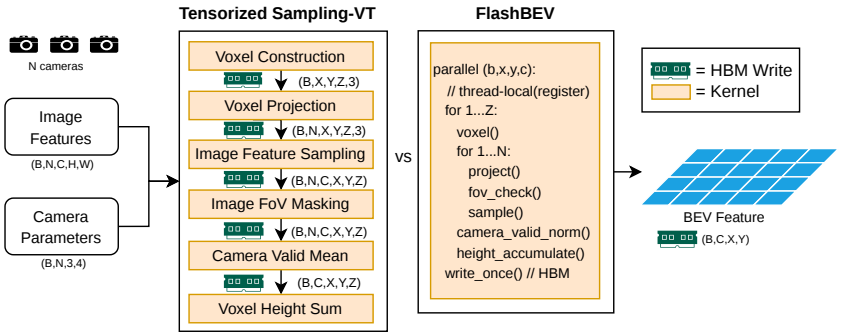

Fig. 2: Left: Tensorized Sampling-VT materializes multiple intermediate tensors in HBM; computation and memory scale as $O(BNCXYZ)$. Right: FlashBEV assigns one thread per (b, x, y, c), performs voxel construction, projection, masking, feature sampling, and reduction locally, and writes only the final BEV feature to global memory.

3.1 Preliminary

We consider a batch size B, N cameras, a BEV grid of size (X, Y), Z height bins, and C feature channels. Sampling-VT aggregates camera features into a BEV feature map $B(x, y, c)$ by computing the *camera-wise valid mean* at each height bin z, then summing over z:

$$B(x,y,c) = \sum_{z=1}^{Z} \frac{\sum_{n=1}^{N} M_n(x,y,z)\, f_n(x,y,z,c)}{\max\left(1, \sum_{n=1}^{N} M_n(x,y,z)\right)}. \tag{1}$$

Here, $f_n(x, y, z, c)$ is the sampled image feature at the projected location, and $M_n(x, y, z) \in \{0, 1\}$ indicates whether voxel (x, y, z) projects to a valid image location for camera n. The mask M_n is 1 when the projected voxel center lies within the camera image with positive depth, and 0 otherwise; f_n is bilinear interpolation from the camera feature map at the projected pixel.

Tensorized Sampling-VT: Baseline We adopt the canonical Sampling-VT formulation (Eq. (1)) and implement it using a fully tensorized execution that explicitly materializes height-dependent intermediate tensors. We refer to this baseline implementation as *Tensorized Sampling-VT*. Prior pipelines (e.g., SimpleBEV) may differ in reduction or implementation details; we use this generic baseline so that efficiency gains are measured consistently. Sec. 4.2 shows that replacing height conv with height sum incurs only a small accuracy drop. Both the baseline and FlashBEV realize Eq. (1) by computing the valid mean over cameras at each height bin z, then summing over z. Algorithm 1 summarizes the baseline execution and its HBM traffic.

Algorithm 1 Tensorized Sampling-VT (Baseline)

Require: Camera features $\mathbf{F}_n$ (N cameras), projection matrices $\mathbf{P} \in \mathbb{R}^{B\times N\times 3\times 4}$; all in HBM. BEV grid (X, Y), height bins Z, channels C.
Voxel construction: build voxel grid; shape $(B, X, Y, Z, 3)$. **Write** voxel grid to HBM.
Voxel projection: project each voxel to each camera; shape $(B, N, X, Y, Z, 3)$. **Write** projected coords to HBM.
Image feature sampling: bilinear sample $\mathbf{F}_n$ at projected locations; Features3D $\in \mathbb{R}^{B\times N\times C\times X\times Y\times Z}$. **Write Features3D to HBM.**
FoV masking: compute validity $M_n(x, y, z) \in \{0, 1\}$ from projected coords; apply mask to Features3D (element-wise). **Write** M_n and masked Features3D to HBM.
Valid camera mean: for each (x, y, z) compute $G(x, y, z, c) = \mathrm{num}_z / \max(1, \mathrm{den}_z)$ with $\mathrm{num}_z = \sum_n M_n f_n$, $\mathrm{den}_z = \sum_n M_n$. **Write** G **to HBM.**
Voxel height sum: $B(x, y, c) \leftarrow \sum_z G(x, y, z, c)$. **Write** B **to HBM.**
return B

3.2 Observations

Intrinsic Memory Bottleneck under Tensorized Execution Under tensorized execution, Sampling-VT materializes sampled 3D tensors whose shapes scale with the height discretization Z, e.g.,

$$\texttt{Features3D} \in \mathbb{R}^{B\times N\times C\times X\times Y\times Z}. \tag{2}$$

These intermediates dominate peak memory and HBM traffic; memory grows linearly with Z, a primary scalability bottleneck. Computation and memory scale as $O(BNCXYZ)$.

Sampling-VT as a Gather–Reduction Operator Despite the memory cost of tensorized execution, Eq. (1) reveals a simple structure: for each BEV location (x, y) and channel c, the output is a sum over height bins z, where at each z a gather–reduction over cameras n yields the valid mean (numerator $\mathrm{num}_z = \sum_n M_n f_n$, denominator $\mathrm{den}_z = \sum_n M_n$, then $\mathrm{num}_z / \max(1, \mathrm{den}_z)$), and $B = \sum_z$ of that. Thus Sampling-VT is a *gather–reduction* operator with per-height-bin normalization. This computation has no cross-(x, y) dependencies, suggesting that each output BEV element can be computed independently using thread-local accumulation. This gather–reduction structure implies that Sampling-VT can be executed as a single fused operator, computing each BEV element independently without staging intermediate tensors.

Splatting-VT requires index sorting and blocks fully thread-local reduction; Sampling-VT's gather pattern admits reordering so each BEV element can be computed independently. We therefore design FlashBEV to exploit this structure; it uses minimal HBM (inputs and output only).

Algorithm 2 FlashBEV

Require: Camera features $\mathbf{F}_n$, projection matrices $\mathbf{P}$; in HBM. Output $B \in \mathbb{R}^{B\times X\times Y\times C}$ in HBM (initialized to zero).
One thread per output (b, x, y, c). All per-thread data kept in registers.
Initialize acc $\leftarrow 0$ (register).
for $z = 1, \ldots, Z$ **do**
  $\mathrm{num}_z \leftarrow 0$, $\mathrm{den}_z \leftarrow 0$
  for $n = 1, \ldots, N$ **do**
    Compute voxel center; project to camera n
    If not valid: **continue**
    $f_n \leftarrow \mathrm{bilinear}(\mathbf{F}_n)$
    $\mathrm{num}_z \leftarrow \mathrm{num}_z + f_n$
    $\mathrm{den}_z \leftarrow \mathrm{den}_z + 1$
  end for
  $g_z \leftarrow \mathrm{num}_z / \max(1, \mathrm{den}_z)$
  acc $\leftarrow$ acc $+ g_z$
end for
Write $B(b, x, y, c) \leftarrow$ acc to HBM once
return B

3.3 FlashBEV

Thread-Local Execution and Reduction Fig. 2 contrasts FlashBEV with Tensorized Sampling-VT. Tensorized execution materializes multiple intermediate tensors in HBM and performs global reductions over (n, z) via global memory. In contrast, FlashBEV launches a fused CUDA kernel over output indices (b, x, y, c), where each thread computes a single BEV element by performing a local reduction over cameras n and height bins z, and writes the result once to global memory.

Concretely, each GPU thread is assigned a BEV location (x, y) and channel c. The thread iterates over height bins z; for each z, it iterates over cameras n, performs projection, validity checking, and bilinear sampling on-the-fly, and accumulates the per-z numerator and denominator ($\mathrm{num}_z \leftarrow \mathrm{num}_z + M_n\, f_n$, $\mathrm{den}_z \leftarrow \mathrm{den}_z + M_n$). After completing the loop over n, the thread sets $G(x, y, z, c) = \mathrm{num}_z / \max(1, \mathrm{den}_z)$. After looping over all z, it writes $B(x, y, c) = \sum_z G(x, y, z, c)$ to HBM. All intermediate quantities exist only transiently in registers and are discarded immediately after use, eliminating intermediate tensor materialization. Algorithm 2 gives the corresponding pseudocode.

Peak memory is $O(BCXY)$, independent of Z and N (Fig. 2).

Operator-Level Equivalence FlashBEV is mathematically equivalent to Tensorized Sampling-VT: the same interpolation and reduction (Eq. (1)) are applied; only the execution order and storage of intermediates change. Inference matches up to floating-point rounding (max error on the order of 10^{-4}, Sec. 4.2); training with FlashBEV yields the same or comparable convergence (Sec. 4.2).

Memory Computation Trade-Off FlashBEV trades increased arithmetic for reduced memory traffic: channel-independent quantities (voxels, projection, validity) are recomputed per channel instead of stored, avoiding $O(BNCXYZ)$ HBM traffic and shifting execution toward compute-bound.

4 Experiments

We evaluate numerical correctness, efficiency, and scalability at the view-transform level under identical network architectures and configurations.

4.1 Experimental Setup

Reference models. We use the SimpleBEV pipeline as the reference BEV architecture: same spatial range and voxelization ($x, y \in [-50, 50]$ m at 0.5 m resolution; $z \in [-5, 5]$ m at 1.25 m resolution), input image size 448×800, backbone downsampling factor 8 (feature map 56×100 per camera), and $C = 128$ channels. Unless otherwise stated, batch size is 1 and the default BEV grid is $X = Y = 200$ with $Z = 8$ height bins; we vary Z, grid size, or C in scaling experiments. Only the view transformation module is replaced by FlashBEV; all other components are unchanged. For detection (baseline validation and E2E) we follow the SimpleBEV training and evaluation protocol; for efficiency we benchmark the view transformation module independently. Tensorized Sampling-VT (our baseline) is implemented as a fully vectorized PyTorch module that materializes the intermediate voxel feature tensor of shape $B \times N \times C \times X \times Y \times Z$; FlashBEV uses fused CUDA kernels and accumulates directly into the BEV grid without intermediate 3D tensors. Under identical configurations both produce numerically equivalent BEV feature maps.

Metrics and protocol. We report inference latency, peak GPU memory, and numerical differences at the BEV feature level. Detection accuracy (IoU) is reported as mean $\pm$ standard deviation over 5 random seeds. All experiments use FP32; we time 100 runs after warm-up with `torch.cuda.synchronize()` and report mean latency; peak memory is `torch.cuda.max_memory_allocated` during the VT forward pass. Profiling on NVIDIA RTX A6000 (multi-GPU results in Fig. 3); training on 8$\times$ NVIDIA H200.

4.2 Accuracy Analysis

We validate correctness at both task level (detection IoU) and operator level (BEV feature and gradient error). Table 1 reports detection IoU on nuScenes [1]. We first reproduce the SimpleBEV baseline (M1, materialized) and implement Tensorized Sampling-VT (M2, materialized) and FlashBEV (M3, fused) within the same pipeline. M2 matches M1 in IoU (46.9 vs. 46.9); M3 (FlashBEV) matches M2 (46.8 vs. 46.9 within variance), preserving detection accuracy with only the execution strategy changed.

Table 1: Accuracy analysis: height reduction and reimplementation (M1, M2), and FlashBEV (M3). Mean ± std over 5 seeds.

ID	Variant	Implementation	Height reduction	IoU
M1	SimpleBEV	Materialized	Conv2d($H{\cdot}C{\to}C$)	**46.9 ± 0.1**
M2	Tensorized Sampling-VT	Materialized	Sum + Conv2d($C{\to}C$)	**46.9 ± 0.3**
M3	FlashBEV (Ours)	Fused	Sum + Conv2d($C{\to}C$)	**46.8 ± 0.1**

Operator-level equivalence. Table 2 reports negligible BEV and gradient errors (mean $\sim 10^{-6}$); Table 1 (M3) confirms equivalent detection accuracy.

Table 2: View transformation (VT) BEV feature and gradient differences; errors negligible (floating-point rounding).

Comparison	Max Error	Mean Error	Mean Rel. Error	Cosine Sim.
Forward	2.93×10^{-4}	8.79×10^{-6}	7.18×10^{-5}	1.00
Backward	1.83×10^{-4}	7.98×10^{-6}	9.22×10^{-6}	1.00

4.3 Efficiency on Reference Config

We use the reference config from Sec. 4.1 (X=Y=200, Z=8, C=128, batch size 1).

E2E performance. We report end-to-end peak memory and latency (Table 3); peak memory decreases from 2211.3 MB to 876.8 MB ($\approx 2.5\times$ lower) and latency from 67.2 ms to 59.9 ms (-7.3 ms, $\approx 11\%$ faster), while detection accuracy remains equivalent. In this reference configuration VT is not the dominant E2E latency component, and we do not decompose per-module latency bottlenecks; the main practical benefit of FlashBEV is *scalability*: larger BEV grids and higher vertical discretization within fixed deployment budgets (Sec. 4.4).

Table 3: End-to-end peak memory and latency for different view transformation (VT) modules. Same architecture and training; only the VT module differs.

Method	E2E Peak Mem (MB)	E2E Latency (ms)
Tensorized Sampling-VT	2211.29	67.19
FlashBEV (Ours)	876.76	59.91

VT forward and backward performance. Table 4 reports VT forward-only and backward-only latency and peak memory: FlashBEV reduces VT peak memory

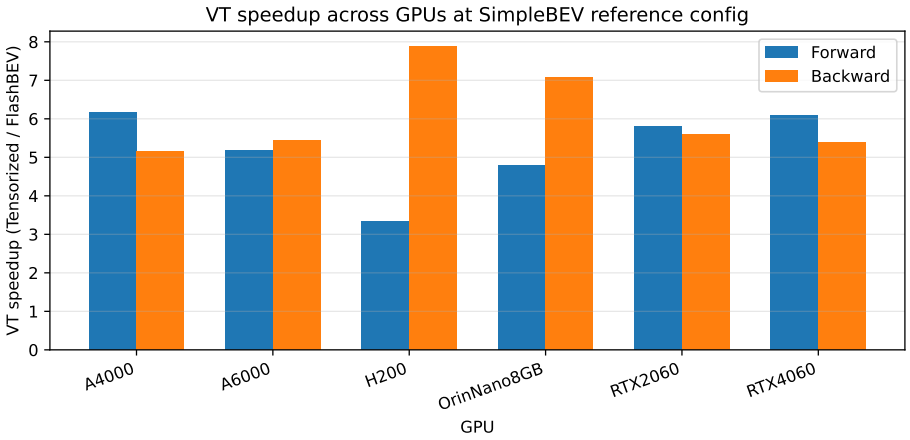

Fig. 3: VT speedup (Tensorized Sampling-VT / FlashBEV) at SimpleBEV reference config (X=Y=200, Z=8, C=128) across six GPUs. FlashBEV achieves forward speedup 3.3–6.2× and backward 5.2–7.9× on all devices.

from 1971.9 MB to 52.8 MB (≈ 37× lower) and achieves about 5.2× forward speedup over Tensorized Sampling-VT.

Table 4: VT forward and backward latency and peak memory (reference config: X=Y=200, Z=8, C=128).

Method	Forward (ms / MB)	Backward (ms / MB)
Tensorized Sampling-VT	9.18 ± 0.23 / 1971.94	17.90 ± 0.20 / 1174.86
FlashBEV (Ours)	1.77 ± 0.03 / 52.82	3.27 ± 0.07 / 105.63

VT speedup across GPUs. To show that gains are not specific to a single GPU, we evaluate VT speedup over Tensorized Sampling-VT on six devices: NVIDIA A4000, A6000, H200, Jetson Orin Nano 8GB, RTX 2060, and RTX 4060, including memory-constrained edge devices (e.g., Jetson Orin Nano 8GB). Fig. 3 reports forward and backward speedup. Across the six GPUs, forward speedup ranges from 3.3× to 6.2× and backward from 5.2× to 7.9×.

4.4 VT Scaling

We analyze how VT memory and latency scale when moving away from the reference config along key design axes (height resolution, BEV grid size, and channel count).

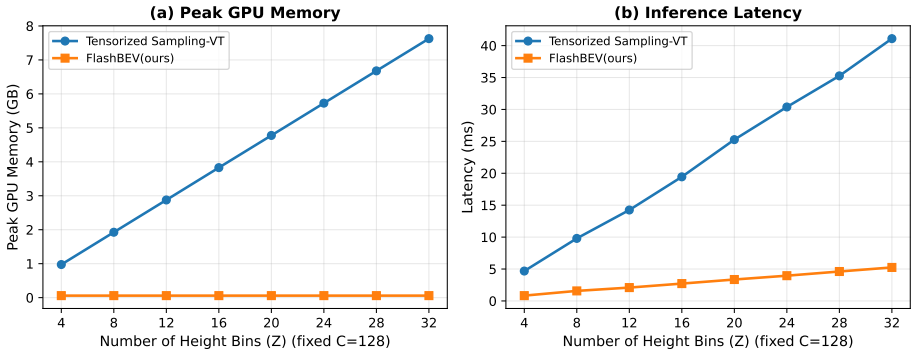

Fig. 4: (a) Peak GPU memory and (b) inference latency vs. number of height bins (Z) with fixed BEV resolution and channels (reference config). Tensorized Sampling-VT exhibits peak memory and latency that grow roughly linearly with Z; FlashBEV achieves substantially lower memory and latency with weak dependence on Z by fusing sampling and reduction into a single kernel.

VT deployment capacity under fixed budgets. We set each budget to Tensorized Sampling-VT's cost at the reference config. Table 5 gives the maximum Z and maximum BEV grid $X{=}Y$ that FlashBEV achieves within that budget; under fixed memory, max Z is N/A because FlashBEV's memory does not scale with Z.

Table 5: Deployment capacity under fixed budgets (latency 9.18 ms or memory 1972 MB at reference config). Max Z is N/A under fixed memory because FlashBEV's memory is independent of Z.

Budget	Max Z	Max BEV grid $X{=}Y$
Fixed latency (9.18 ms)	58 (7.25×)	529 (2.65×)
Fixed memory (1972 MB)	N/A	1992 (10×)

VT memory and latency vs. height bins. As shown in Fig. 4, Tensorized Sampling-VT exhibits near-linear growth in peak memory usage and latency as Z increases, due to explicit materialization of intermediate 3D tensors whose size scales with $O(XYZ)$. FlashBEV maintains a substantially lower memory footprint that is independent of Z, and a much lower latency that grows far more slowly with Z than the tensorized baseline, since it avoids height-dependent intermediates and accumulates within a fused kernel.

BEV grid scaling. We vary the BEV grid size (X, Y) (longer range or finer resolution) and report the maximum grid achievable under fixed deployment

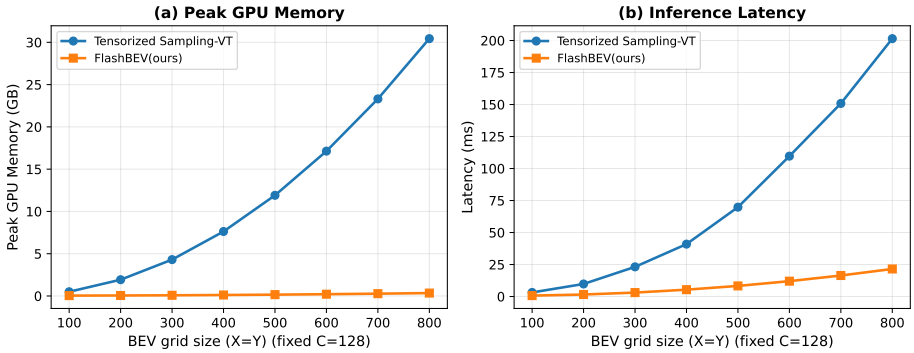

Fig. 5: (a) Peak GPU memory and (b) inference latency vs. BEV grid size (X=Y) with fixed height bins and channels (reference config). Tensorized Sampling-VT exhibits strong growth in memory and latency with grid size; FlashBEV maintains much lower memory and latency, enabling larger grids under the same budget.

constraints. Fig. 5 shows how memory and latency scale with BEV grid size; Table 5 gives deployment capacity under both budget types.

Channel dimension scaling. FlashBEV trades increased arithmetic for reduced memory traffic: channel-independent quantities (voxels, projection, validity) are recomputed per channel rather than stored as large tensors. Fixing the BEV grid (X=Y=200) and height bins (Z=8), we sweep C (Fig. 6). For typical C (128–256) FlashBEV keeps a substantial speedup; for larger C recomputation cost grows and the relative speedup can narrow, but FlashBEV still avoids the OOM failures of the baseline.

4.5 Deployment in TensorRT and Cross-Framework Applicability

The preceding results use SimpleBEV in PyTorch; we now show the same fused, exact execution deploys through a standard inference toolchain and transfers to a different VT operator. All TensorRT measurements in this section use an NVIDIA RTX 4070 Ti SUPER (FP32), a different device from the RTX A6000 used above, so TRT numbers should be compared within each table rather than against the PyTorch results.

Deployment in TensorRT. TensorRT [23] is a widely used inference engine for deploying autonomous-driving perception, and recent BEV methods explicitly target it [4]. We implement FlashBEV as a TensorRT plugin and compare against a Tensorized Sampling-VT plugin under FP32 (Table 6): peak memory stays constant as height bins grow from Z=8 to Z=32 while the tensorized baseline grows with Z, and FlashBEV remains numerically exact (zero max error).

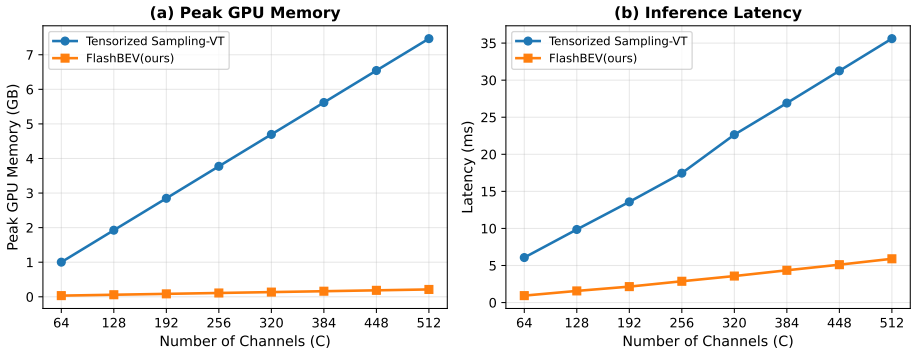

Fig. 6: (a) Peak GPU memory and (b) inference latency vs. number of channels (C); reference config. FlashBEV achieves lower memory and high speedup for typical C; for very large C speedup can narrow but FlashBEV remains faster with much lower memory.

Table 6: FlashBEV as a TensorRT plugin vs. Tensorized Sampling-VT (FP32), sweeping height bins Z. Peak memory (external delta) in MB, latency in ms; × is Tensorized/FlashBEV. FlashBEV's memory is independent of Z and it remains numerically exact at all Z (zero max error; cf. Table 2).

	Peak mem (MB)			Latency (ms)		
Z	Tensorized	FlashBEV	×	Tensorized	FlashBEV	×
8	1294	**38**	34	5.622	**1.128**	5.0
16	2514	**38**	66	20.132	**2.140**	9.4
32	4994	**38**	131	40.193	**4.177**	9.6

Cross-framework applicability: BEVFormer SCA. To test whether this execution idea is specific to SimpleBEV-style Sampling-VT, we apply it to the spatial cross-attention (SCA) of BEVFormer [17], a deformable-attention view transformation. Our kernel, **FlashSCA**, performs on-the-fly projection with thread-local recomputation so the per-camera BEV-shaped intermediates are never materialized in HBM; the baseline mirrors the dense per-camera SCA path used in public TensorRT deployments. FlashSCA reproduces BEVFormerSCA's output and gradient up to FP32 round-off ($\max|\Delta|=7\times10^{-7}$ forward, 7×10^{-6} backward), so it is an exact, training-compatible replacement requiring no retraining. At BEVFormer-base defaults (N=6, C=256, 8 heads, BEV 200×200, Z=4, L=4) we sweep the number of sampling points P in TensorRT (Table 7): FlashSCA reduces peak memory by 2.5–3.2× and latency by 1.7–2.1×, with gains widening in P because BEVFormerSCA materializes a sampling-points tensor of shape $[N{\cdot}X{\cdot}Y, H, L, P, 2]$ that FlashSCA never builds.

Table 7: FlashSCA vs. BEVFormerSCA in TensorRT (FP32) at BEVFormer-base defaults, sweeping sampling points P. Peak memory in MB, latency in ms; × is BEVFormerSCA/FlashSCA.

	Peak mem (MB)			Latency (ms)		
P	BEVFormerSCA	FlashSCA	×	BEVFormerSCA	FlashSCA	×
8	1178	**464**	2.54	8.96	**5.27**	1.70
16	1990	**698**	2.85	15.81	**8.16**	1.94
32	3708	**1168**	3.18	29.27	**13.90**	2.11

5 Conclusion

Our contribution is an *execution design* for sampling-based view transformation (Sampling-VT): we recast it as a gather–reduction operator and propose **FlashBEV**, a single fused, IO-aware kernel that preserves exact operator-level equivalence while substantially reducing memory traffic and kernel-launch overhead. FlashBEV reduces VT peak memory by over 37× and achieves 5.2× forward speedup (Table 4) with memory independent of height bins; it also deploys in TensorRT and transfers to BEVFormer's SCA (FlashSCA), opening a new design space for scalable BEV perception on commodity and edge devices.

Acknowledgements

The authors thank Prof. Keiji Kimura of Waseda University for insightful discussions.

References

1. Caesar, H., Bankiti, V., Lang, A.H., Vora, S., Liong, V.E., Xu, Q., Krishnan, A., Pan, Y., Baldan, G., Beijbom, O.: nuscenes: A multimodal dataset for autonomous driving. IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) pp. 11618–11628 (2020)
2. Chambon, L., Zablocki, É., Chen, M., Bartoccioni, F., Pérez, P., Cord, M.: Pointbev: A sparse approach to bev predictions. In: IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR). pp. 15195–15204 (2024)
3. Chen, T., Moreau, T., Jiang, Z., Zheng, L., Yan, E.Q., Shen, H., Cowan, M., Wang, L., Hu, Y., Ceze, L., Guestrin, C., Krishnamurthy, A.: Tvm: An automated end-to-end optimizing compiler for deep learning. In: OSDI (2018)
4. Chen, Y., Song, H., Tao, W., Mo, S., Zhang, S., Hua, X., Zhao, T.: Fastbev++: Fast by algorithm, deployable by design. arXiv preprint arXiv:2512.08237 (2025)
5. Chi, X., Liu, J., Lu, M., Zhang, R., Wang, Z., Guo, Y., Zhang, S.: Bev-san: Accurate bev 3d object detection via slice attention networks. In: IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR). pp. 17461–17470 (2023)

6. Chu, X., Deng, J., You, G., Duan, Y., Li, Y., Zhang, Y.: Rayformer: Improving query-based multi-camera 3d object detection via ray-centric strategies. In: ACM International Conference on Multimedia (MM) (2024)
7. Dao, T., Fu, D., Ermon, S., Rudra, A., Ré, C.: Flashattention: Fast and memory-efficient exact attention with io-awareness. Advances in neural information processing systems **35**, 16344–16359 (2022)
8. Harley, A.W., Fang, Z., Li, J., Ambrus, R., Fragkiadaki, K.: Simple-BEV: What really matters for multi-sensor BEV perception? In: IEEE International Conference on Robotics and Automation (ICRA). IEEE (2023)
9. Hu, H., Wang, F., Su, J., Wang, Y., Hu, L., Fang, W., Xu, J., Zhang, Z.: Ea-lss: Edge-aware lift-splat-shot framework for 3d bev object detection. arXiv preprint arXiv:2303.17895 (2023)
10. Huang, J., Huang, G.: Bevdet4d: Exploit temporal cues in multi-camera 3d object detection. arXiv preprint **abs/2203.17054** (2022)
11. Huang, J., Huang, G.: Bevpoolv2: A cutting-edge implementation of bevdet toward deployment. arXiv preprint arXiv:2211.17111 (2022)
12. Huang, J., Huang, G., Zhu, Z., Ye, Y., Du, D.: Bevdet: High-performance multi-camera 3d object detection in bird-eye-view. arXiv preprint **abs/2112.11790** (2021)
13. Jiang, Y., Zhang, L., Miao, Z., Zhu, X., Gao, J., Hu, W., Jiang, Y.G.: Polarformer: Multi-camera 3d object detection with polar transformers. In: AAAI Conference on Artificial Intelligence (2023)
14. Li, P., Shen, W., Huang, Q., Cui, D.: Dualbev: Unifying dual view transformation with probabilistic correspondences. In: European Conference on Computer Vision (ECCV) (2024)
15. Li, Y., Huang, B., Chen, Z., Cui, Y., Liang, F., Shen, M., Liu, F., Xie, E., Sheng, L., Ouyang, W., et al.: Fast-bev: A fast and strong bird's-eye view perception baseline. arXiv preprint arXiv:2301.12511 (2023)
16. Li, Y., Ge, Z., Yu, G., Yang, J., Wang, Z., Shi, Y., Sun, J., Li, Z.: Bevdepth: Acquisition of reliable depth for multi-view 3d object detection. In: AAAI Conference on Artificial Intelligence (AAAI) (2023)
17. Li, Z., Wang, W., Li, H., Xie, E., Sima, C., Lu, T., Qiao, Y., Dai, J.: BEVFormer: Learning Bird's-Eye-View Representation from Multi-Camera Images via Spatiotemporal Transformers. In: European Conference on Computer Vision (ECCV) (2022)
18. Li, Z., Yu, Z., Wang, W., Anandkumar, A., Lu, T., Alvarez, J.M.: Fb-bev: Bev representation from forward-backward view transformations. In: IEEE/CVF International Conference on Computer Vision (ICCV). pp. 6896–6905 (2023)
19. Liu, H., Teng, Y., Lu, T., Wang, H., Wang, L.: Sparsebev: High-performance sparse 3d object detection from multi-camera videos. In: IEEE/CVF International Conference on Computer Vision (ICCV) (2023)
20. Liu, Z., Tang, H., Amini, A., Yang, X., Mao, H., Rus, D., Han, S.: Bevfusion: Multi-task multi-sensor fusion with unified bird's-eye view representation. In: IEEE International Conference on Robotics and Automation (ICRA) (2023)
21. Lu, S.W., Tsai, Y.H., Chen, Y.T.: Toward real-world bev perception: Depth uncertainty estimation via gaussian splatting. In: IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR). pp. 17124–17133 (2025)
22. Ma, W., Jiang, J., Yang, Y., Chen, Z., Chen, H.: Lssinst: Improving geometric modeling in lss-based bev perception with instance representation. In: International Conference on 3D Vision (3DV) (2025)

23. Nvidia tensorrt. NVIDIA Developer, https://developer.nvidia.com/tensorrt (2024), accessed: 2026-06-28
24. Philion, J., Fidler, S.: Lift, splat, shoot: Encoding images from arbitrary camera rigs by implicitly unprojecting to 3d. In: European Conference on Computer Vision (ECCV). pp. 194–210. Springer (2020)
25. Reading, C., Harakeh, A., Chae, J., Waslander, S.L.: Categorical depth distribution network for monocular 3d object detection. In: IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) (2021)
26. Wu, D., Yang, F., Xu, B., Liao, P., Zhao, W., Zhang, D.: Hv-bev: Decoupling horizontal and vertical feature sampling for multi-view 3d object detection. IEEE Transactions on Intelligent Transportation Systems **26**(11), 18734–18746 (2025)
27. Wu, Y., Li, R., Qin, Z., Zhao, X., Li, X.: Heightformer: Explicit height modeling without extra data for camera-only 3d object detection in bird's eye view. IEEE Transactions on Image Processing **34**, 689–700 (2025)
28. Yang, C., Chen, Y., Tian, H., Tao, C., Zhu, X., Zhang, Z., Huang, G., Li, H., Qiao, Y., Lu, L., Zhou, J., Dai, J.: Bevformer v2: Adapting modern image backbones to bird's-eye-view recognition via perspective supervision. In: IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) (2023)
29. Yang, J., Xie, E., Liu, M., Alvarez, J.M.: Parametric depth based feature representation learning for object detection and segmentation in bird's-eye view. In: IEEE/CVF International Conference on Computer Vision (ICCV). pp. 8483–8492 (October 2023)
30. Yang, L., Yu, K., Tang, T., Li, J., Yuan, K., Wang, L., Zhang, X., Chen, P.: Bevheight: A robust framework for vision-based roadside 3d object detection. In: IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) (2023)
31. Zhu, X., Su, W., Lu, L., Li, B., Wang, X., Dai, J.: Deformable detr: Deformable transformers for end-to-end object detection. In: International Conference on Learning Representations (ICLR) (2021)