Real-Time BEV-Based Occupancy Perception for AMR via Knowledge Distillation

Shiang Yu Tsai[1] and Yu-Cheng Chang[1]

Department of Mechanical Engineering, National Taiwan University
r13522859@ntu.edu.tw
r13522804@ntu.edu.tw

Abstract. The goal of our project is to build a real-time perception module deployable on embedded platforms that directly supports navigation and obstacle avoidance. The module outputs both an occupancy map and a semantic map. To satisfy low-latency and low-compute constraints, we incorporate temporal modeling to better represent dynamic objects (e.g., pedestrians and vehicles) while maintaining stable estimates of static structures.

In the early stage, we adopted a Gaussian-based streaming occupancy baseline and attempted to reduce voxelization cost by shrinking the observation range and cropping occupancy ground truth (GT). Profiling, however, revealed that the backbone and the data-processing pipeline remain largely fixed costs. As a result, even though voxelization can be locally accelerated by approximately 22×, the overall end-to-end speedup is still bounded by Amdahl's law, and we only observed an approximate 2.6× overall speedup in practice. This suggests that voxel-range shrinkage alone is unlikely to overcome the bottleneck.

Therefore, we shifted to a low-compute Bird's-Eye-View (BEV)-based student design to reduce the cost of the entire inference chain (backbone + pipeline + head). We employ a high-accuracy occupancy model as the teacher model for knowledge distillation. Our goal is to recover BEV representation accuracy under strict efficiency and latency constraints, ultimately delivering a navigation-friendly perception module that can stably output traversable regions and dynamic risk information for real-world AMR applications.

Keywords: Occupancy Perception · BEV Representation · Knowledge Distillation

1 Progress Summary

1.1 Limitations of Gaussian-Based Streaming Occupancy for Real-Time Use

At the beginning, we adopted a Gaussian-based streaming occupancy approach (GaussianWorld) as our first baseline. The key motivation was its support for temporal fusion, while the GaussianFormer family (GaussianFormer, GaussianFormer-2, and GaussianWorld) also provides a clean Gaussian scene representation for

3D semantic occupancy. To push this baseline toward real-time performance, we implemented occupancy GT cropping and shrank the observation range, with the goal of reducing the voxelization and rendering workload.

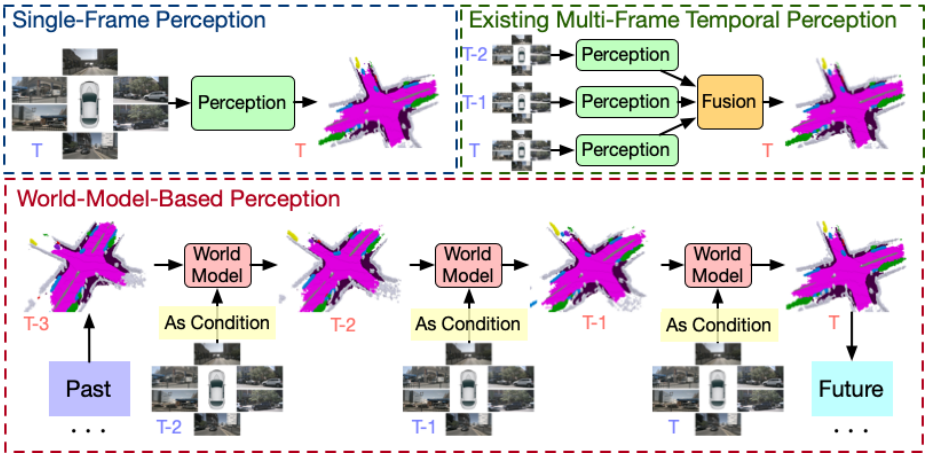

Fig. 1. GaussianWorld pipeline visualization. We tried shrinking the observation range and GT cropping, but end-to-end speedup was still limited by fixed costs (backbone + data pipeline).

Investigation through profiling confirmed that shrinking the voxel-related computation yields a clear *local* speedup. However, the end-to-end runtime remained largely dominated by the backbone network and the data-processing pipeline, which effectively act as fixed costs. As a result, even with substantially faster voxelization, the overall speedup is still limited, making it difficult to meet real-time constraints on our target platform.

1.2 Transition to a BEV-Based Student with a Framework-Independent PyTorch Pipeline

Based on the profiling results, we shifted to a BEV-based student direction. The motivation is straightforward: we must reduce not only voxel-related computation, but also the overall cost of the backbone network and the end-to-end training/inference pipeline.

In practice, many 3D occupancy projects are strongly coupled to the OpenMMLab ecosystem (mmcv/mmengine). A large portion of those public repositories assumes mmcv 1.x for training runners and data pipelines. Meanwhile, our GPU environment (RTX 5090) requires a newer PyTorch stack (PyTorch 2.7.1), which quickly introduced compatibility issues and substantial maintenance overhead if we continued relying on an mmcv-style pipeline.

We initially attempted a hybrid approach: retaining the mmcv dataloader/pipeline while rewriting only the model and inference in pure PyTorch. However, matching our environment required upgrading mmcv to 2.x, and we found that mmcv 1.x (which mainstream occupancy methods utilize) and 2.x differ substantially in runners, training APIs, and pipeline design. Because these components are tightly coupled, a "partial migration" (detaching only the necessary parts) was difficult in practice. After several hours of migration attempts, we decided to stop fighting the framework and instead rewrote the training pipeline and dataloader entirely in pure PyTorch.

To move beyond this bottleneck, we needed a representation that reduces computation across the entire inference chain rather than accelerating a single module. A BEV-based formulation is a natural choice under this constraint: it keeps most feature processing in 2D/BEV (which is significantly cheaper than dense 3D feature encoding) and outputs a planning-friendly intermediate representation that is directly useful for navigation. In this way, the workload can be reduced not only in voxelization/rendering but also in the backbone-to-head pipeline, which is exactly what the profiling results suggested was necessary.

Aside from runtime issues, the BEV formulation is also practical for our workflow: it is easier to compress compute/memory than a dense 3D voxel head, it provides a clean target for distilling teacher BEV representations (features/logits), and it makes deployment (e.g., ONNX/TensorRT) as well as integration to navigation costmaps more straightforward.

For the BEV-based design, we primarily referenced FlashOcc for the overall workflow: multi-camera images → 2D features → BEV features → BEV occupancy head, followed by lifting to 3D via a cheap Channel-to-Height reshaping. The most challenging part was the dataloader, since occupancy prediction is highly sensitive to alignment, augmentation order, label formatting, and geometric consistency.

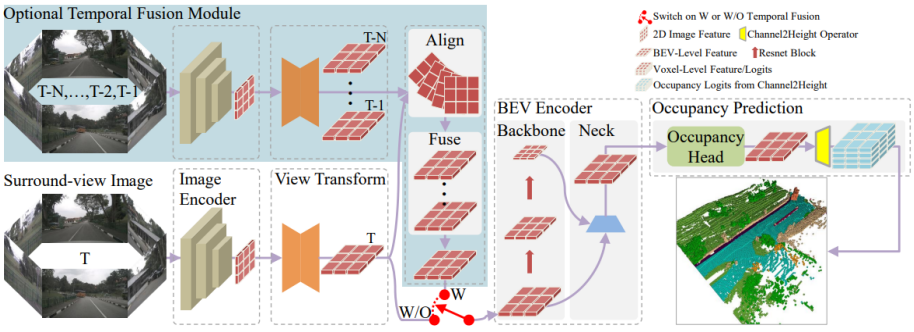

Fig. 2. FlashOcc-style BEV student pipeline visualization. We switched to a lighter end-to-end chain and rewrote training + dataloader in pure PyTorch.

Many implementation details that were previously encapsulated within mmcv pipelines had to be made explicit. We therefore referenced a GaussianFormer-style dataloader logic and rebuilt multi-view augmentation and label packaging step by step. After multiple rounds of debugging and verification, we eventually obtained a stable pure PyTorch dataloader that can run training reliably, which became the foundation for our subsequent BEV student training and distillation.

1.3 Teacher Choice and Knowledge Distillation Design

After committing to the BEV-student and pure-PyTorch direction, we required a clear strategy for (1) teacher selection and (2) the specific distillation targets, so that the student can recover accuracy while remaining computationally efficient.

We retained a Gaussian-based teacher line to maintain consistency with our initial GaussianWorld baseline: GaussianWorld adopts the GaussianFormer-style Gaussian scene representation and supports temporal fusion, making the GaussianFormer family a natural candidate space for teacher models. However, a common limitation of early Gaussian-style approaches is the *random seeding* initialization scheme: Gaussians are first placed randomly and then iteratively moved toward occupied regions via gradient-based optimization. In practice, this may lead to early-stage missed occupancy and slow convergence. The issue is further amplified when probabilistic Gaussians are expected to represent *only* occupied regions: if many Gaussians are initialized in free space, subsequent refinement may not fully "catch up" to recover missed occupied structures.

Therefore, we chose **GaussianFormer-2** as the teacher. Its key improvement is a *distribution-based initialization module*. Instead of relying solely on random initialization, it leverages camera geometry to sample multiple depth reference points along each pixel ray and queries GT occupancy at these points to construct a pixel-aligned occupancy distribution as supervision. This encourages image features to learn the *entire occupancy distribution along the ray* (rather than only a single surface depth), thereby initializing Gaussians closer to truly occupied regions. This choice is also consistent with our BEV-student philosophy: first obtain a reliable intermediate representation, and then produce efficient downstream outputs.

Two aligned supervisions from the teacher. GaussianFormer-2 naturally provides two aligned supervision signals: (1) voxelized 3D occupancy on a 3D voxel grid (capturing full 3D geometry and semantics), and (2) a BEV representation obtained by collapsing the 3D occupancy along the height axis (planning-friendly and serving as an efficient 2D intermediate signal).

Student design (FlashOcc-style). Our student follows a FlashOcc-like efficient pipeline: multi-camera images $\rightarrow$ 2D features $\rightarrow$ BEV features (**distill point 1**) $\rightarrow$ BEV occupancy head $\rightarrow$ Channel-to-Height reshaping $\rightarrow$ 3D voxel logits (**distill point 2**). The core idea is to avoid expensive 3D convolutions and 3D feature encoders, let 2D/BEV features carry most of the computation, and lift to 3D via a cheap channel rearrangement.

Distill point 1: BEV feature distillation. **Goal:** align the student's BEV intermediate representation with the teacher such that geometry and semantics remain consistent, and mitigate the common failure mode where training loss becomes low but validation does not improve due to misalignment. **How:** we perform distillation at the BEV feature level. The teacher provides BEV supervision (either a teacher BEV feature or a BEV map collapsed from 3D occupancy), and the student provides its BEV feature for matching. To prevent large empty/background regions from diluting the supervision signal, we adopt a masked strategy: we generate a mask from GT occupancy and apply distillation primarily on occupied or near-occupied regions. If teacher and student feature channels do not match, we introduce a lightweight alignment module (e.g., a 1×1 convolution or an MLP) before computing the distillation loss.

Distill point 2: 3D logit distillation after Channel-to-Height. Because the student produces 3D logits via reshaping (efficient but potentially fragile for height-wise semantics), we further apply teacher-guided 3D supervision to preserve the global 3D distribution. Specifically, we distill the 3D voxel logits using channel-wise logit distillation (CWD): each semantic channel is treated as a distribution over the 3D voxel space, and teacher–student alignment is enforced via KL divergence. This is typically more stable than voxel-wise matching because it aligns the global spatial distribution per class.

Overall loss.

$$\mathcal{L}_{total} = \mathcal{L}_{stu} + \mathcal{L}_{depth} + \lambda_{feat}\mathcal{L}_{MFD} + \lambda_{logit}\mathcal{L}_{CWD}.$$

In short: BEV distillation aligns intermediate geometry/semantics first, and 3D logit distillation aligns the final 3D semantic distribution. The two distillation points are complementary.

1.4 Teacher Dump and Distillation Data Flow

To enable knowledge distillation, we export the teacher outputs from GaussianFormer-2 into offline numpy files and load them in the student dataloader for aligned supervision.

1. **Teacher signals.** We use two teacher predictions from GaussianFormer-2 for distillation: (i) a 3D voxel probability tensor of size $200 \times 200 \times 16 \times 18$, and (ii) a height-collapsed BEV probability tensor of size $200 \times 200 \times 18$.
2. **Offline dumping to `.npy`.** The teacher outputs are first dumped into offline numpy files (`.npy/.npz`), as illustrated in Fig. 3, which provides a stable and reproducible supervision source for student training.
3. **Loading teacher supervision in the dataloader.** During training, our dataloader loads the dumped teacher tensors together with the corresponding sample data, and uses them to compute distillation supervision with correct sample-level alignment.

4. **Current status.** The end-to-end distillation pipeline (teacher dump → dataloader load → KD training) has been successfully executed, as shown in Fig. 4. We have implemented an initial mIoU evaluation, but the current computation may be incorrect or not fully consistent with the mainstream occupancy evaluation protocol. Due to time constraints, we have not yet completed a standard-aligned mIoU verification, RayIoU evaluation, nor summarized the final qualitative results.

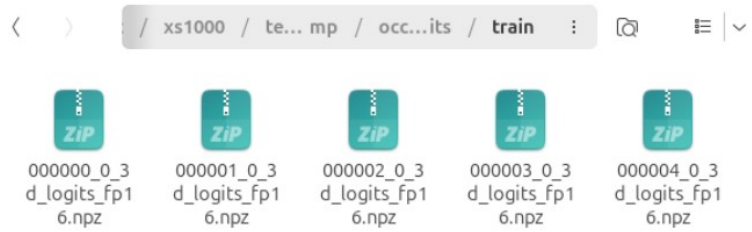

Fig. 3. distallation npz

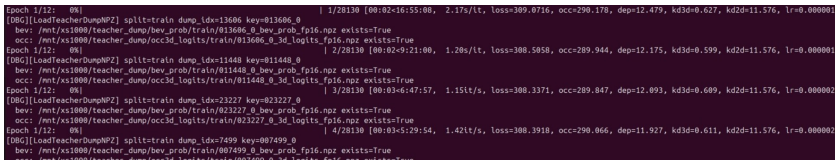

Fig. 4. training log

2 What We Did

We carried out the project in three stages.

Stage 1: Gaussian-based baseline and profiling. We first established GaussianWorld as a streaming occupancy baseline and implemented observation-range shrinkage and occupancy GT cropping to reduce voxelization/rendering cost. We then profiled the end-to-end runtime and confirmed an Amdahl-style bottleneck: even with a large local speedup in voxel-related computation, the overall latency remained dominated by the backbone and the data-processing pipeline.

Stage 2: Framework-independent BEV student pipeline. Motivated by the profiling results, we transitioned to a FlashOcc-style BEV student design

to reduce computation across the entire inference chain (backbone + pipeline + head). Due to incompatibilities between our PyTorch stack (PyTorch 2.7.1) and mmcv-based training pipelines, we rewrote the training/inference workflow in pure PyTorch, including training scripts, logging/validation, and a geometry-consistent multi-view dataloader rebuilt by referencing GaussianFormer-style data processing (augmentation order, label packaging, and camera geometry bookkeeping).

Stage 3: Teacher integration and knowledge distillation. We selected GaussianFormer-2 as the teacher and implemented a two-branch distillation scheme: BEV-level supervision for aligning intermediate BEV representations and 3D logit-level supervision (after Channel-to-Height lifting) for stabilizing the final 3D semantic distribution. To make training reproducible and efficient, we dumped teacher predictions offline (3D: $200 \times 200 \times 16 \times 18$, BEV: $200 \times 200 \times 18$) into `.npy/.npz` files and loaded them in the student dataloader for aligned KD supervision. The end-to-end KD training pipeline is now running successfully, and we implemented an initial mIoU evaluation, but it was not finalized due to possible mismatches with mainstream evaluation protocols.

3 Problems and Fixes

3.1 Performance Bottleneck: Amdahl Limit

Problem: Even if voxelization can be accelerated significantly by shrinking the observation range (local speedup around 22×), the end-to-end speedup is still limited by fixed costs from backbone and data pipeline. This makes it hard to reach real-time; in our experiments the overall speedup was only around 2.6×.

Fix: We switched to a BEV-based student design to reduce computation across the entire inference chain, instead of optimizing only one module.

3.2 Framework Compatibility: mmcv 1.x vs 2.x

Problem: We initially wanted to keep mmcv dataloader/pipeline and only rewrite model inference in pure PyTorch. However, our environment required PyTorch 2.7.1, and the matching mmcv version required moving to mmcv 2.x. In practice, mmcv 1.x and 2.x differ a lot in runner, training APIs, and pipeline design, and components are highly coupled, making partial migration unrealistic.

Fix: We stopped retrofitting the old framework and rewrote the entire training pipeline in pure PyTorch. This removed dependency conflicts and made the codebase easier to control.

3.3 Dataloader Difficulty: Alignment, Augmentation, and Geometry Consistency

Problem: Occupancy prediction is extremely sensitive to alignment and geometry consistency. After removing mmcv pipelines, many hidden assumptions

became explicit tasks: augmentation order, multi-view packaging, label formatting, and consistent geometry inputs (camera intrinsics/extrinsics, transforms) used downstream. A small mismatch can lead to a typical failure mode: training loss goes down but validation stays poor.

Fix: We referenced GaussianFormer dataloader logic and rebuilt the PyTorch dataloader step by step: (1) make augmentation deterministic/reproducible and check intermediate outputs; (2) verify tensor shapes/types at each step; (3) sample a few cases for visualization/metadata comparison to ensure geometry and labels are consistent.

Fig. 5. Before alignment/geometry fixes: BEV student outputs were very poor and often collapsed to predicting mostly empty space.

Fig. 6. After alignment and geometry-consistency fixes: outputs became much more reasonable and started to capture occupancy structure.

3.4 Loading the Teacher Model Was Difficult

Problem: Loading the teacher (GaussianFormer-2) was not straightforward in practice. There were dependency requirements, checkpoint key mismatches, and strict expectations on preprocessing and geometry inputs. Small inconsistencies in camera parameters, transforms, or tensor formatting could break inference or produce unstable supervision.

Fix: We reduced scope first: make the teacher forward reliably on a small fixed sample set and confirm it can output the two aligned supervisions we need (3D occupancy and BEV-collapsed BEV supervision). Then we incrementally aligned keys/shapes/outputs (key mapping checks + shape verification) before integrating it into the distillation loop.

3.5 Missing MMCV Ops: Pool Plugin Replacement and Conv Rewrites

Problem: Because we moved away from the mmcv/mmengine ecosystem, some FlashOcc components could not be used directly. In particular, we did not use the FlashOcc pooling plugin (which is often integrated via MMCV custom ops). In addition, many repositories rely on mmcv-style layers and wrappers (e.g., certain conv blocks or utility modules), and replacing them with pure PyTorch required extra re-implementation work.

Fix: We replaced the pooling/plugin-dependent parts with pure PyTorch alternatives and rewrote the required layers/modules explicitly. The key was to keep the pipeline behavior consistent (input/output shapes, geometry assumptions) while minimizing custom dependencies.

3.6 Training Crashes Mid-Run (Over One Week of Debugging)

Problem: For more than a week, training often crashed in the middle of a run. Our dataset was stored on an external drive. Initially, system logs suggested an I/O-related issue, so we tried changing mount settings (including NTFS mounting), but the crashes still happened. We confirmed that GPU RAM and CPU RAM were not the bottleneck. We also tried lowering dataloader workers and limiting thread usage, but the instability remained. Eventually, we suspected the OS/kernel environment might have been affected by other installations in the lab (e.g., Isaac Sim installation touching kernel/driver components).

Fix: We reinstalled the OS to reset the environment. After that, the crashing issue disappeared. For other lab members, using a Docker-based isolated environment also avoided the crash, suggesting the root cause was likely OS/kernel environment instability rather than model logic.

3.7 Teacher Dump to `.npy` and Dataloader Alignment

Problem: To perform knowledge distillation efficiently, we needed to use GaussianFormer-2 teacher outputs as supervision during student training. However, directly running the teacher online inside the student training loop was impractical due to heavy dependencies, strict input formatting requirements, and high runtime overhead. In addition, ensuring sample-level alignment between the dumped teacher tensors and the student inputs (images/geometry/GT) was non-trivial: even a small mismatch in indexing, augmentation order, or tensor layout could silently corrupt the distillation signal.

Fix: We adopted an offline dumping strategy. We exported the teacher predictions into `.npy/.npz` files (3D: $200 \times 200 \times 16 \times 18$, BEV: $200 \times 200 \times 18$) and modified the student dataloader to load these teacher tensors together with the corresponding sample data. We added sanity checks for tensor shapes, keys, and sample correspondence, ensuring that the distillation supervision is correctly aligned with the student training samples.

3.8 mIoU Evaluation Was Not Fully Resolved

Problem: We implemented an initial mIoU evaluation for voxel-level occupancy. However, the computed mIoU values appeared inconsistent with expected trends, suggesting that our implementation may deviate from the mainstream occupancy evaluation protocol (e.g., class indexing, ignore/free handling, masking rules, or voxel aggregation details). Due to limited time, we were not able to fully diagnose and reconcile these differences, and thus the evaluation results were not finalized.

Fix (ongoing): We plan to cross-check our mIoU computation against standard evaluation code and definitions, verify label conventions (including *free* and *ignore* treatment), and add small-scale unit tests (toy examples with known confusion matrices) to validate correctness before reporting final numbers. In the next stage, we will also add RayIoU and class-wise IoU for a more complete evaluation.

4 Future Work

- **Align and complete evaluation.** We have implemented an initial mIoU evaluation, but its definition/implementation may differ from the mainstream occupancy evaluation protocol. We will align our metric computation to standard practice via cross-checking and add complementary metrics (RayIoU and class-wise IoU) for a complete quantitative comparison.
- **Tune the distillation strategy.** We will systematically tune distillation weights (e.g., λ_{feat}, λ_{logit}) and study masking/target choices through ablations to improve the accuracy–efficiency trade-off.
- **Strengthen temporal modeling for dynamic objects.** We will incorporate lightweight temporal aggregation and/or temporal consistency constraints to reduce flickering and improve the representation of dynamic objects (e.g., pedestrians and vehicles).
- **Embedded benchmarking and navigation integration.** We will benchmark latency/FPS/memory under embedded-like settings (e.g., smaller inputs, FP16/INT8) and validate navigation-friendliness by integrating the BEV occupancy outputs into the costmap/planning stack.

References

1. GaussianWorld: Gaussian World Model for Streaming 3D Occupancy Prediction. CVPR (2025)
2. GaussianFlowOcc: Sparse and Weakly Supervised Occupancy Estimation Using Gaussian Splatting and Temporal Flow. arXiv (2025)
3. GaussianFormer: Scene as Gaussians for Vision-Based 3D Semantic Occupancy Prediction. ECCV (2024)
4. GaussianFormer-2: Probabilistic Gaussian Superposition for Efficient 3D Occupancy Prediction. CVPR (2025)
5. Improvements in Multiprocessor System Design. ACM SIGARCH Computer Architecture News (1985)

6. FlashOcc: Fast and Memory-Efficient Occupancy Prediction via Channel-to-Height Plugin. arXiv (2023)
7. Occ3D: A Large-Scale 3D Occupancy Prediction Benchmark for Autonomous Driving. NeurIPS (2023)
8. Efficient Vision-Based Occupancy Prediction with Knowledge Distillation. Pattern Recognition (2025)