Camera-only 3D Semantic Occupancy on Waymo with OccFormer

Su Hyun Kim (SJSU ID: 018219422)
Team 9, Option 2 (Fine-Tune & Train)

December 15, 2025

Abstract

This project began as a fork of the public OccFormer codebase and ended as a working Waymo occupancy pipeline inside MMDetection3D: dataset loading, training/evaluation scripts, and a qualitative video demo. Most of the work was integration and debugging rather than changing the core model. The main friction points were aligning Waymo KITTI-format sensor exports with Occ3D-Waymo occupancy labels, fixing label conventions (free-space label), dealing with the fact that Waymo GT volumes are commonly 200×200×16 while the default model grid is 256×256×32, and keeping training stable on a single 12 GB GPU. The best run that completed end-to-end in this repo is `baseline_fast` (10% data via `load_interval=10`, EfficientNet-B7, CrossEntropy), which reached 13.41 mIoU on the validation set based on the printed IoU table in `results/baseline_fast/logs/evaluate.log`. Additional experiments included stronger augmentation, learning-rate changes, SGD, and multiple imbalance variants (focal loss, weighted CE, fewer Mask2Former queries). Several runs failed for concrete engineering reasons (registry import order, wrong image loader, SLURM environment issues, and CUDA out-of-memory), and this report documents those failures with log excerpts and the specific files modified.

1 Introduction & Problem Description

Semantic occupancy represents a driving scene as a dense 3D voxel grid around the ego vehicle, where each voxel is assigned an occupancy state and a semantic class label. Unlike box-based perception, this representation explicitly models both free space and scene structure, which is directly useful for downstream planning and remains informative under partial occlusions.

This project studies OccFormer [1], a recent camera-only 3D semantic occupancy model. The core objective is to fine-tune OccFormer on Waymo occupancy data and evaluate the resulting model with reproducible training and testing procedures. Rather than proposing a new architecture, the primary contribution is building an end-to-end, repeatable pipeline that makes a public occupancy model run on a new large-scale dataset under practical compute constraints.

Option 2 (fine-tune/train) was chosen because the main learning outcome was understanding the model and training pipeline in depth, and then making the full training/evaluation workflow run reliably in an HPC environment.

2 Background / Related Work

OccFormer [1] is a camera-only 3D semantic occupancy model built around a "lift to 3D, then reason in 3D" design. Given multi-view RGB images, it extracts per-view features using a standard image backbone and neck. To recover 3D structure without LiDAR at inference time, it predicts depth distributions and applies a Lift-Splat-Shoot style view transformation to project image features into a voxel-aligned 3D volume in the ego coordinate system. This step connects 2D pixels to 3D voxels and provides geometry-aware features for subsequent 3D reasoning.

To remain efficient on large voxel grids, OccFormer avoids fully dense attention and instead uses a two-pathway transformer encoder: a local pathway that attends within 3D windows for fine-grained structure, and a global pathway that propagates long-range context through a BEV-style representation. The decoder adapts Mask2Former to 3D occupancy: a fixed set of queries predicts class scores and mask embeddings, which are combined with voxel features to produce voxel-level logits. Figure 1 summarizes the overall framework.

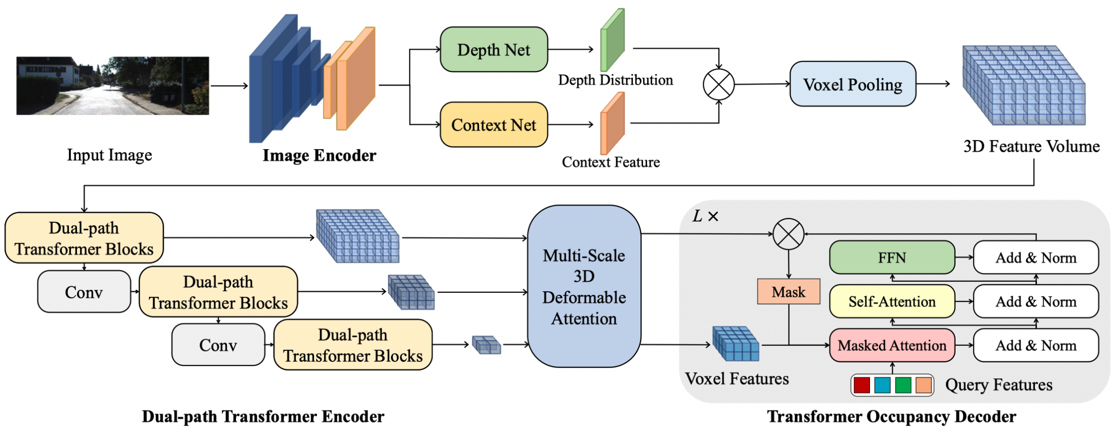

Figure 1: OccFormer framework.

A common challenge in occupancy learning is severe class imbalance, since most voxels correspond to free (empty) space. Focal loss [2] is widely used to emphasize rare and hard examples, and focal-loss-based training was explored as one of the configurations. In practice, however, the additional memory cost during backpropagation limited stability in the available training environment, so the final results prioritize configurations that could complete reliably end-to-end.

3 System Design and Implementation Details

3.1 System environment

OccFormer is designed to run in a Python 3.7 environment with `cudatoolkit=11.3`. However, the class HPC cluster did not provide CUDA 11.3 as a supported system module. To match the original dependency stack, external CUDA 11.3 wheels were downloaded and a compatible environment was built around them, including the corresponding versions of Python, `mmcv-full`, `mmdetection`, and `mmdetection3d`.

All experiments were executed on the HPC GPU nodes. The allocated GPU varied by partition and job availability, but most training and evaluation runs were conducted on a Tesla P100 (see the `nvidia-smi` header in `results/waymo_full/waymo_full_15649.out`).

3.2 Repository layout

Before getting into dataset details, it helps to see how the repository is organized, because most of the engineering work in this project was making these pieces fit together. The overall layout follows

Component	Version
Python	3.7.16
PyTorch	1.10.1 + CUDA 11.3.1
MMCV-Full	1.4.0
MMDetection	2.14.0
MMDetection3D	0.17.1
GPU	Tesla P100-PCIE-12GB

Table 1: Software/hardware stack used for this project.

the standard MMDetection3D project structure (model code as a plugin + configs + entry-point scripts), and the Waymo-specific parts are concentrated in a small number of folders:

```
OccFormerWithWaymoData/
  mmdetection3d/             # upstream framework (MMDet3D)
  projects/
    mmdet3d_plugin/          # OccFormer + datasets/losses/pipelines
    configs/occformer_waymo/# Waymo configs + experiment presets
  tools/                     # runnable entry points
  scripts/                   # SLURM wrappers and quick checks
  data/                      # Waymo data, exports, occupancy GT
  results/                   # experiment outputs
  reports/final_report/      # LaTeX report + figures
```

From the original OccFormer fork, the main additions in this class project are: (1) a Waymo experiment runner that applies preset overrides, (2) Waymo utilities for keeping metadata consistent, (3) a video demo generator, and (4) a small set of Waymo-specific dataset/pipeline hooks.

3.3 Data: Waymo sensor exports + Occ3D-Waymo occupancy labels

This project combines Waymo sensor exports (images + LiDAR) and Occ3D-Waymo occupancy ground truth. These two sources must agree on frame indexing and camera geometry. Occ3D-Waymo labels here are voxelized at 0.4 m resolution over a range of approximately $[-40, -40, -1, 40, 40, 5.4]$ meters, which corresponds to a $200\times200\times16$ volume [5]. The semantic labels are 0–14, and free space is stored as label 23.

In the HPC environment, Waymo data is stored under a single versioned root, and the three subtrees below are the ones the code relies on:

```
data/waymo_v1-3-1/
  waymo_format/      # raw TFRecords from Waymo Open Dataset
  kitti_format/      # KITTI-style export generated by MMDet3D converter
  occ3d_waymo/       # Occ3D-Waymo GT (.npz) + info/pose PKLs
```

Waymo sensor data (images + LiDAR). Waymo Open Dataset v1.3.1 TFRecords are converted into a file-based KITTI-format layout using the built-in MMDetection3D converter (`create_data.py waymo`). One practical detail is that the converter depends on TensorFlow; when launched on a CPU-only node it can print CUDA-related warnings (missing `libcuda.so.1`) even though the conversion proceeds on CPU.

Occupancy ground truth (Occ3D-Waymo). Occ3D-Waymo provides per-frame voxel ground truth stored as `.npz` files, plus metadata pickles that describe the frame list and camera poses. Because partial downloads and missing frames are common in an HPC workflow, the metadata files are filtered to keep only samples that exist on disk and match the converted sensor exports.

One practical detail that mattered for reproducibility is that the Waymo conversion tool relies on TensorFlow. When run on a CPU-only node, it can print CUDA-related warnings (missing `libcuda.so.1`) even though the conversion proceeds on CPU. Because metadata and local file layout can drift (missing files, different roots, partial downloads), filtered info files were generated using `tools/filter_waymo_infos.py`, which scans the KITTI-format directory, drops samples with missing sensor or occupancy files, and can optionally subsample.

3.4 Waymo dataset interface and model configuration

The original OccFormer codebase already contained a Waymo dataset class, but it was designed around a KITTI-style 3D detection workflow: it focuses on boxes and LiDAR-based calibration files, and it does not naturally carry the Occ3D occupancy voxel labels or the camera-pose metadata that OccFormer needs for image-to-voxel lifting. In other words, "loading Waymo" is not enough for this project: the training loop needs a dataset interface that (1) feeds OccFormer the multi-view camera geometry in the expected format and (2) provides voxel ground truth for occupancy supervision.

To make that work, the Waymo occupancy training loop is built around `CustomWaymoDataset_T` (a wrapper on top of `CustomWaymoDataset`). The key changes are practical rather than algorithmic: it reads per-frame, per-camera pose information (intrinsics and extrinsics) from the Occ3D-Waymo metadata, assembles the multi-view camera inputs that the OccFormer pipeline expects, and attaches occupancy ground truth as `gt_occ` so the Mask2Former occupancy head can compute voxel losses.

On the model side, the Waymo config keeps the original OccFormer architecture (image backbone + neck, view transformer, 3D encoder, Mask2Former-style occupancy head). An experiment-preset layer and a dedicated training entry point (`experiments.py` + `train_waymo.py`) are used for iteration speed, so controlled changes (optimizer/LR, augmentations, backbone size, query count, and loss choice) can be rerun without duplicating config files or mixing outputs across runs.

3.5 End-to-end pipeline overview

Figure 2 shows a overall high-level architecture diagram that matches the implementation. The two data sources (Waymo sensor exports and Occ3D voxel labels/poses) go through the dataset/pipeline interface that assembles multi-view inputs and occupancy GT, and the OccFormer model runs with the data, ending with losses/metrics and artifacts (checkpoints, logs, demo video).

3.6 Engineering challenges and resolutions

Most effort in this project was spent making a large research codebase run reliably on a new dataset under realistic cluster constraints. The model architecture itself was largely kept intact; progress depended on removing integration blockers and making the training/evaluation loop robust to common failure modes (missing files, mismatched conventions, SLURM environment differences, and GPU memory limits).

1) nuScenes-style vs Waymo-style loader mismatch OccFormer originates from nuScenes/SemanticKITTI code paths where some pipelines assume nuScenes-style nested dictionaries. Waymo KITTI-format exports use a flatter structure. When the wrong loader was used, training

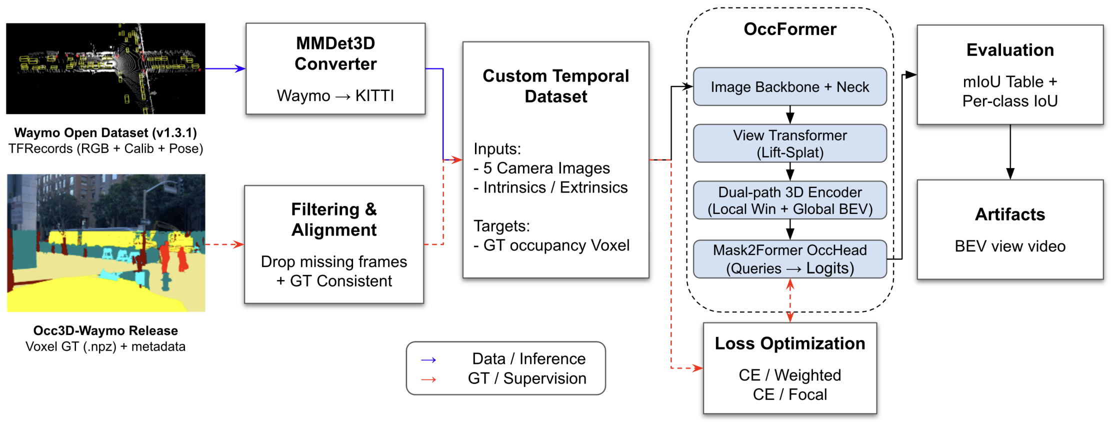

Figure 2: Overall architecture of the Waymo occupancy pipeline built in this project (data alignment → dataset/pipeline → OccFormer inference/training → evaluation and artifacts).

failed immediately. The resolution was to switch the data pipeline to the Waymo/KITTI-style multi-view loader so that image tensors and metadata are assembled in the expected format, and the result was making a new custom dataset loader.

2) Camera ordering and pose alignment A silent but damaging issue occurred when the camera pose ordering from the Occ3D metadata did not match the on-disk camera folder naming for two views. This does not necessarily crash training, but it corrupts geometry because the view transformer projects features using the wrong intrinsics/extrinsics. The resolution was to correct the two-view swap consistently both in sample construction and in the metadata alignment step, so pose indices and image paths remain consistent end-to-end.

3) Label convention mismatch (free space) Occ3D-Waymo occupancy labels include a free-space label stored as 23, while the training/evaluation label set in this repository treats free space as class 15. Without remapping, training targets and evaluation classes do not match and splitted out error at evaluation step. It was simple to fix, just change label 23 to 15, but took a long time to realize the problem.

4) Voxel grid shape mismatch (GT vs model output) Waymo occupancy ground truth commonly uses a $200\times200\times16$ voxel grid, while many OccFormer configurations and visualization assumptions expect $256\times256\times32$. This produced shape mismatch failures in loss computation and also broke visualization when comparing GT and predictions. The resolution was to implement an explicit nearest-neighbor resizing option for GT at load time (controlled by experiment presets), and to defensively align shapes in the demo renderer when visualizing predictions against GT.

5) GPU memory limits during query-based decoding On a 12 GB GPU, the most frequent hard blocker was CUDA out-of-memory in the query-based occupancy head. The root cause is a large intermediate tensor with shape `[B,Q,H,W,D]` that is created during mask-to-voxel logit composition, and its memory footprint grows linearly with the number of queries Q. To mitigate this, the experiments progressively reduced the query count in memory-heavy presets, switched to smaller backbones when necessary, and implemented more memory-aware focal-loss variants. In practice, the debugging workflow was iterative: adjust a preset, reproduce the OOM failure, and verify whether peak memory dropped enough to finish training. The most reliable setting found was $Q = 30$, which eliminated OOM in the target configuration, but there was not enough time to complete the full experiment schedule (one run typically takes about two days).

4 Evaluation & Testing Results

4.1 Experimental details

Before presenting quantitative results, Table 2 summarizes the fine-tuning setups used in this repository. All runs follow the same overall task definition (camera-only semantic occupancy on a fixed voxel grid) and use the Waymo training split for optimization and the Waymo validation split for evaluation. Most experiments intentionally use a subsampled dataset via `load_interval`=10 to make iteration feasible on limited GPU resources; the remaining changes focus on learning-rate/optimizer choice, backbone size, loss function for class imbalance, query count, and (for some runs) the voxel-grid and GT resizing policy.

Experiment	Backbone	Loss	Opt	LR	Epochs	Q	Grid	GT resize	In (H×W)	Load int.	SpG	Val int.
baseline	EfficientNet-B7	CrossEntropy	AdamW	1.0e-04	30	100	256×256×32	-	640×960	10	1	0
lr_5e4	EfficientNet-B7	CrossEntropy	AdamW	5.0e-04	30	100	256×256×32	-	640×960	10	1	0
sgd	EfficientNet-B7	CrossEntropy	SGD	1.0e-02	30	100	256×256×32	-	640×960	10	1	0
strong_aug	EfficientNet-B7	CrossEntropy	AdamW	1.0e-04	35	100	256×256×32	-	640×960	10	1	0
improved_imbalance_q30	EfficientNet-B4	Weighted CE	AdamW	1.0e-04	30	30	256×256×32	-	640×960	10	1	0

Table 2: Summary of experiment presets used for Waymo fine-tuning in this repository. The default baseline uses EfficientNet-B7, AdamW (1e-4), CrossEntropy, 100 queries, and a 256×256×32 grid. Most runs use `load_interval`=10 for faster iteration.

4.2 Evaluation metrics

Semantic mIoU over occupancy classes is used as the primary metric. Per-class IoU tables are read from the printed summary at the end of each `results/*/logs/evaluate.log`. One confusing detail is that the progress messages in these logs still say "Evaluating nuScenes occupancy" even for Waymo; this string comes from reused evaluation code.

Another detail that matters for reproducibility: the evaluation logs print a correct per-class IoU table, but then end with showing 0.0 in `mIoU`. This report uses the printed per-class IoU table (which includes mIoU) as the quantitative result, because the returned metric dict is clearly inconsistent.

4.3 nuScenes sanity check (baseline still works)

Although the main target dataset for this project was Waymo, a nuScenes "sanity check" was kept in the loop. The goal was to confirm that the original OccFormer nuScenes path still runs and produces reasonable metrics before and after Waymo-specific changes (custom dataset class, pipeline edits, plugin imports), rather than failing due to a broken environment or a registry/import regression.

The log `logs/nusc_infos.11975.out` records one such run using the released nuScenes checkpoint `ckpts/occformer_nusc_r50.pth`. At the end of the run, the log prints the nuScenes LiDAR segmentation mean IoU (0.692) and the occupancy metrics (SC IoU 0.130, SSC mIoU 0.102), along with a full per-class IoU table for the 16 lidarseg classes.

Metric (nuScenes)	Value
LiDARSeg mean IoU	0.692
Occupancy SC IoU	0.130
Occupancy SSC mIoU	0.102

Table 3: nuScenes sanity-check metrics printed in `logs/nusc_infos.11975.out`.

4.4 Waymo validation results (mIoU and per-class IoU)

Waymo validation results were consolidated into `final_report/iou_result.csv` for reporting. Not every preset in Table 2 produced a complete evaluation run (several stopped early due to integration or memory issues), so Table 4 reports the experiments that are included in the consolidated CSV (and reserves rows for any additional runs listed in `final_report/report_experiments.txt`).

Experiment	mIoU	general_object	vehicle	pedestrian	sign	cyclist	traffic_light	pole	construction_cone	bicycle	motorcycle	building	vegetation	tree_trunk	road	walkable	free
baseline_fast	13.41	0.00	18.69	9.61	1.79	7.22	0.68	4.34	4.05	1.04	0.00	8.08	7.78	3.52	35.56	20.10	92.02
strong_aug	12.43	0.00	18.36	9.62	1.99	3.34	1.72	3.96	4.21	0.01	0.00	8.33	7.57	3.13	26.19	18.98	91.46
sgd	11.97	0.00	16.88	7.79	1.40	1.01	0.01	3.40	0.13	0.00	0.00	7.37	5.46	2.32	33.92	19.60	92.29
lr_5e4	7.29	0.00	0.58	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	18.77	4.75	92.56
improved_imbalance_q30	pending	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-

Table 4: Waymo validation results (IoU in %) consolidated from `iou_result.csv`. Rows are experiments and columns are per-class IoU, with mIoU shown first.

Result visualization Figure 3 visualizes the same per-class IoU values in a form that makes cross-experiment differences easier to see.

4.5 Analysis of Waymo results

Overall performance level The reported mIoU values (Table 4) are low compared to typical camera-based occupancy results on curated benchmarks. In this repository, the main goal was to make the full Waymo pipeline run end-to-end under tight compute limits, and most experiments were intentionally run with `load_interval`=10 (approximately 10% of the available training frames) to keep turnaround time feasible on a single 12 GB GPU. Under that constraint, the model tends to underfit rare classes and learns a strong bias toward the dominant classes.

Class imbalance dominates the error profile The strongest signal in Figure 3 is the extreme gap between frequent and rare classes. Free space stays above 91 IoU in every completed run, while several rare classes remain near 0 IoU (for example, `general_object` and `motorcycle` are 0 across all reported experiments). Because mIoU averages over classes, a few near-zero classes significantly reduce the final score even when common structure classes are learned reasonably well. This behavior is consistent with occupancy supervision where most voxels are free space and small objects occupy very few voxels.

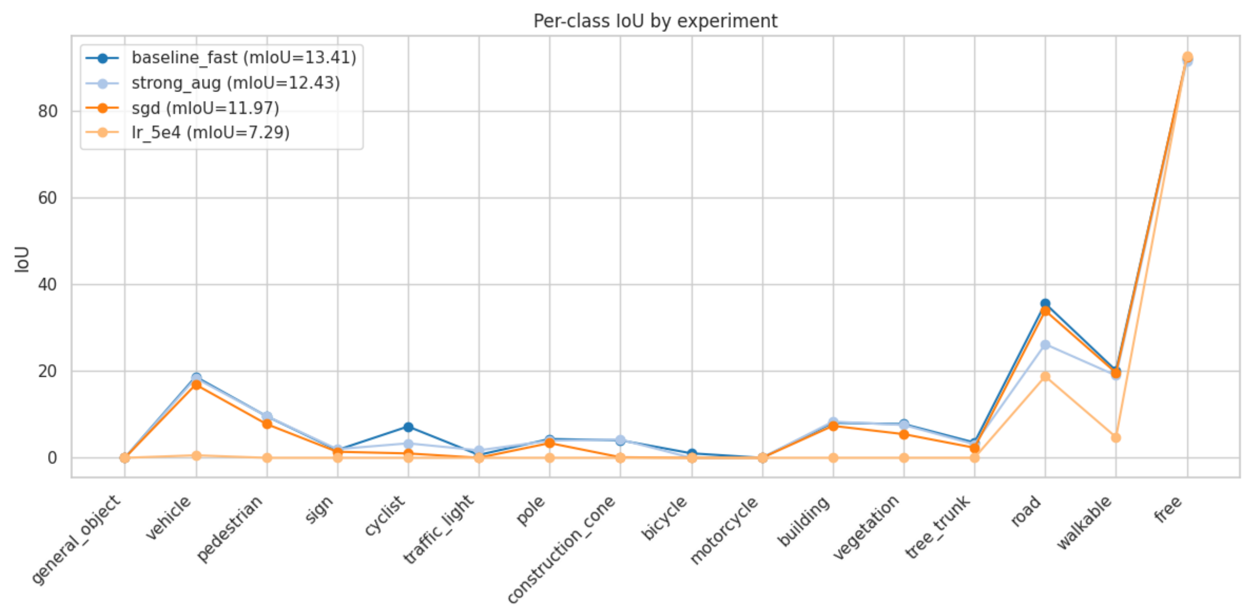

Figure 3: Per-class IoU across experiments (from `final_report/iou_result.csv`).

Geometry-sensitive task vs aggressive augmentation Comparing `baseline` and `strong_aug`, stronger augmentation does not improve mIoU and tends to reduce some geometry-heavy classes (for example, `road` drops from 35.56 to 26.19, and `cyclist` drops from 7.22 to 3.34). In camera-only occupancy, the view transformer relies on accurate image-to-voxel geometry; overly aggressive photometric or geometric perturbations can make learning harder when the dataset is already subsampled and the model is not fully converged.

4.6 Visualize

This project adds `tools/make_waymo_demo.py` script to generate an OccFormer-style video for Waymo that shows a camera mosaic and BEV occupancy slices from both prediction and GT:

```
python tools/make_waymo_demo.py \
  projects/configs/occformer_waymo/waymo_base.py \
  results/baseline_fast/model/latest.pth \
  --scene-id 0 --max-frames 120 --slice-idx 8 --fps 5 \
  --out results/waymo_demo/baseline_fast_scene0.mp4
```

The script imports the plugin so custom datasets are registered, unwraps MMCV `DataContainer` objects for visualization, and resizes GT when its grid shape does not match the model output. One example of the video result can be found here.

4.7 Reproducibility

To rerun the experiment, go to SJSU HPC and follow the instruction

```
srun -p gpuqs --pty /bin/bash
conda activate occformer307
cd /home/018219422/OccFormerWithWaymoData
export PYTHONPATH=$(pwd):$PYTHONPATH
```

Training with SLURM wrappers:

```
sbatch scripts/run_experiment.sh baseline_fast
```

Additional runs use the same wrapper with a different preset name from Table 2:

```
sbatch scripts/run_experiment.sh improved_imbalance_q30
```

Evaluation:

```
python tools/test.py projects/configs/occformer_waymo/waymo_base.py \
  results/baseline_fast/model/latest.pth --eval mIoU --launcher none
```

5 Conclusion

This project demonstrates an end-to-end, reproducible pipeline for fine-tuning a public camera-only 3D semantic occupancy model (OccFormer) on Waymo occupancy labels (Occ3D-Waymo) under practical HPC constraints. The main outcome is not a new model architecture, but a working system that aligns Waymo sensor exports with voxel ground truth, supports training/evaluation through stable entry-point scripts, and produces both quantitative results and a qualitative demo video.

Quantitatively, the best completed run in this repository (`baseline`) achieved 13.41 mIoU on the Waymo validation split while training under a single 12 GB GPU budget with a reduced data fraction. The experiments also clarified several engineering-critical factors for occupancy learning on Waymo: label convention consistency (free-space remapping), GT grid size mismatches ($200\times200\times16$ vs $256\times256\times32$), and the dominant memory bottleneck in Mask2Former-style query decoding due to large `[B,Q,H,W,D]` intermediates.

The primary limitations were GPU memory headroom and experiment time, which restricted the ability to complete longer schedules and to fully evaluate more memory-intensive imbalance strategies (e.g., focal loss with larger backbones). In addition, the evaluation pipeline currently prints correct per-class IoU tables but returns an inconsistent metric dictionary in some runs, which should be corrected for cleaner reporting.

Future work will focus on running the full configuration sweep on larger GPUs (including query-count and focal-loss variants), increasing the training data fraction once a stable memory baseline is established, and fixing the evaluation output so the final reported metric is consistent across logs and returned summaries.

References

[1] Y. Zhang, Z. Zhu, and D. Du, "OccFormer: Dual-path Transformer for Vision-based 3D Semantic Occupancy Prediction," arXiv:2304.05316, 2023.

[2] T.-Y. Lin, P. Goyal, R. Girshick, K. He, and P. Dollár, "Focal Loss for Dense Object Detection," ICCV 2017.

[3] Waymo Open Dataset, https://waymo.com/open/.

[4] OpenMMLab MMDetection3D, https://github.com/open-mmlab/mmdetection3d.

[5] X. Tian, T. Jiang, L. Yun, Y. Mao, H. Yang, Y. Wang, Y. Wang, and H. Zhao, “Occ3D: A Large-Scale 3D Occupancy Prediction Benchmark for Autonomous Driving,” *Advances in Neural Information Processing Systems*, vol. 36, pp. 64318–64330, 2023.