Occupancy Flow Prediction Report

As part of Samsung Worklet ID-MSRIT5ELITE our project name is "**Occupancy Flow Prediction for Automotive Vision**"

Problem Statement :

- To develop a predictive model that leverages agents' tracks from the past 1-second data on a specific map to forecast the bird's-eye view (BEV) occupancy and traffic flow for both currently-observable and obstructed vehicles.

- The model should accurately project the status of these vehicles for the upcoming 8-second duration, taking into account various potential scenarios and accounting for any potential occlusions during this time period.

Occupancy Flow Fields (OFF) for Motion Forecasting in Autonomous Driving

Introduction:

Previous methods suffered from the problems of:

1. Limited Overall Motion Capture
2. Loss of Individual Object Tracking
3. Lack of Directional Information

Hence, OFF has been proposed which contributes:

1. Occupancy Flow Fields
2. Flow-Traced Occupancy
3. Speculative Agents

Implementation:

OFF uses two key components to predict future movements:

1. **Occupancy Grid:** This is a grid-like map representing the road ahead. Each cell in the grid contains a value between 0 and 1, indicating the probability of a vehicle or pedestrian occupying that space at a future time.
2. **Flow Field:** This is also a grid map, but with arrows in each cell. The arrow points to the location where the object in that cell likely came from in the previous moment. By analysing these "backward flows," the model can predict where the object might be going next based on its recent movement.

Architecture

The OFF model receives several inputs to understand the current traffic situation:

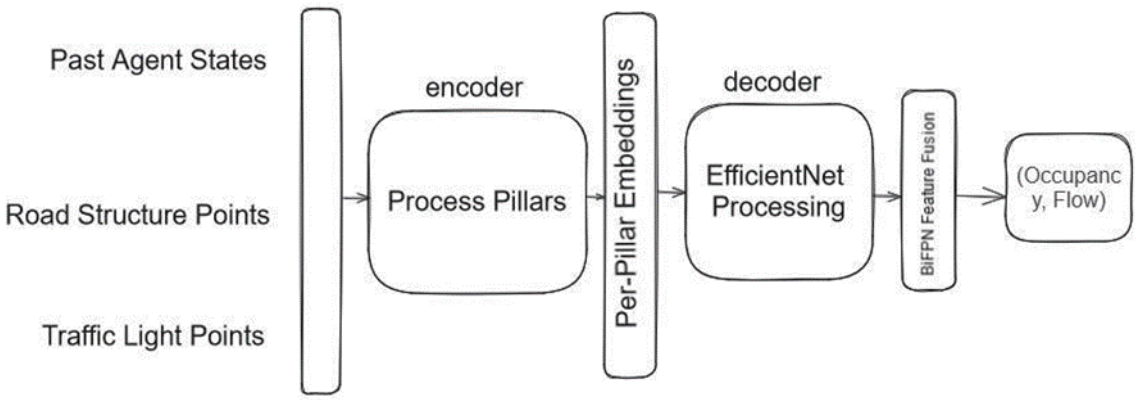
- **Past Agent States (At):** This includes information about previously detected vehicles and pedestrians, including:
  - Position (x, y coordinates)
  - Orientation (angle)
    - Dimensions (width, height)
    - Velocity (speed and direction) for moving objects
- **Road Structure:** This provides details about the road layout:
  - Line segments and curves representing lanes and traffic markings
  - Type of each road element (crosswalk, stop sign, lane marker, etc.)
- **State of Traffic Lights:** This provides real-time information for each traffic light the car might encounter:
  - Position of the traffic light
  - Current state (red, yellow, green)

The model outputs a prediction for each grid cell, including:

- Whether a pedestrian or vehicle will be present in that cell at a future time.
- The predicted movement (direction and speed) of the object in that cell.

The OFF model consists of two main parts:

- **Encoder:** This processes the input data (past positions, road layout, traffic lights). It includes:
    - PointPillars-like processing to convert 3D data into a 2D grid.
    - Agent state encoding to represent past positions with additional information like time, speed, and direction.
    - Pillar feature extraction to extract relevant features from the data.
- **Decoder:** This interprets the compressed information from the encoder and predicts future occupancy and flow:
    - Per-pillar embeddings: Receives the compressed data from the encoder.
    - EfficientNet-like processing: Analyses the data using a model similar to EfficientDet, which is good at object recognition in images.
    - BiFPN feature fusion: Combines different levels of detail from the EfficientNet model to incorporate both specific object locations and broader context like road layout.
    - Occupancy and flow prediction: Predicts the probability of a pedestrian or vehicle being in a specific location in the future and their movement (direction and speed).



<u>Results:</u>

Advantages:

1. **Benefits of Separate Predictions**: The model predicts separate occupancy grids and flow fields for vehicles and pedestrians to improve accuracy due to their different movement patterns.
2. **Backward Flow Advantage**: Unlike traditional methods that predict future movement directions, OFF uses "backward flow" arrows. This simplifies calculations and allows for multiple possible future paths for an object since one backward flow can represent several past locations.

<u>Evaluation Metrics:</u>

- AUC (Area Under the Curve): 0.365
- SoftIoU (Intersection over Union): 0.196

Challenges:

- **Unseen Objects:** Current observations might not capture everything (e.g., a car entering from a side street).
- **Solution: Speculative Prediction:** The model incorporates a "speculative prediction model" to predict movements of these unseen objects. It uses the same structure as the main OFF model but focuses on predicting occupancy and flow for these future agents.
- **Training with Alternative Labels:** The model is trained using "alternative labels" that provide information about future agents not present in the past data. This helps the model learn patterns and predict movements of unseen objects.

Implicit Occupancy Flow Fields for Perception and Prediction in Self-Driving

Introduction:

Previous methods suffered from the problems of:

1. miss objects due to limitations in sensor range or occlusions.
2. Inefficient as it involves separate steps for detection and prediction, leading to potential inaccuracies
3. High Computational Cost
4. Handling of Irrelevant Areas

New Approach:

1. **Flexibility**: ImplicitO is not limited by predefined grids, enabling versatile predictions at any point in space and future time.
2. **Efficiency**: Focuses on relevant areas for the motion planner, reducing computational load and optimizing resource use.
3. **Backward Flow Prediction**

Training Dataset:

- Argoverse 2 Sensor (AV2): 700 sequences of U.S. city data with LiDAR, maps, and annotations.
- HighwaySim: Simulated dataset with Pandar64 LiDAR and lane graphs.

Testing Dataset:

- AV2: 150 sequences reserved for validation/testing.

Implementation:

- The model integrates directional information for precise analysis and decision-making in autonomous driving.
- It addresses unseen object challenges with speculative prediction models and enhances training with alternative labels to improve learning and prediction capabilities.

Input Data:

- **LiDAR Data**: Sequence of LiDAR sweeps with points represented by (px, py) coordinates relative to the self-driving car and height (ph) above the ground.
- **HD Map Data**: Raster image displaying lane centerlines (C).

Output Data:

- **Occupancy (o)**: Probability (0 to 1) that a point in space will be occupied by an object in the future.
- **Flow (f)**: 2D vector indicating the backward motion (in BEV) of an object occupying that point from current to previous time step.

Architecture:

- **Encoder**: Processes LiDAR data and HD map to create a feature map (Z) containing scene geometry, semantics, and motion information.
- **Decoder**: Utilises the feature map (Z) and query points (Q) to predict occupancy and flow for each query point. It leverages the idea that distant objects can influence occupancy at a query point.

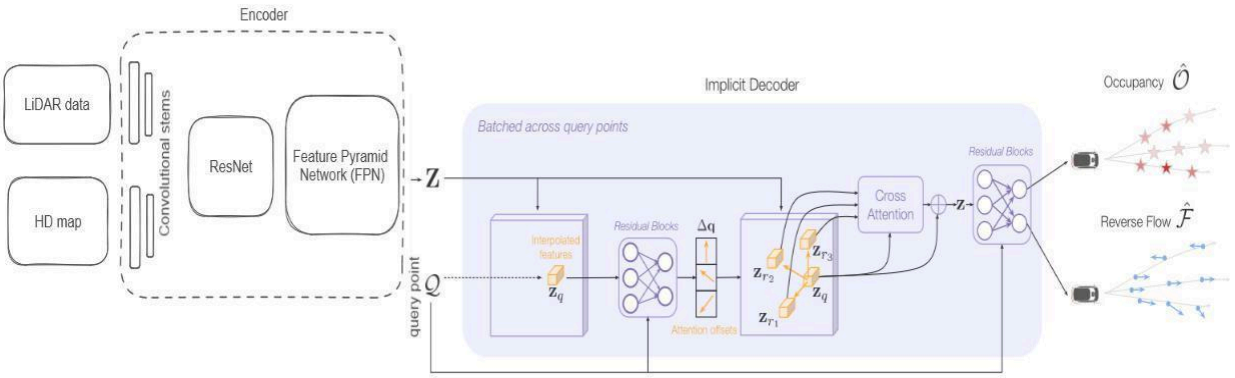
Encoder Details:

1. **Convolutional Stems**:
    - Two stems for LiDAR and map data.
2. **ResNet**:
    - Extracts features from combined data at different resolutions.
3. **Feature Pyramid Network (FPN)**:

- Processes features from ResNet to produce feature map (Z) with half the resolution of input data, capturing spatial and temporal information.

Decoder Details:

1. **Feature for Query Point**:
   - Bilinear interpolation extracts a feature vector (zq) from feature map (Z) at the query point's location, capturing local information.
2. **Reference Points and Features**:
   - Predicts K reference points (r1 to rK) using learned offsets (Δq).
   - Deformable convolution-like mechanism focuses on specific areas of the feature map.
   - Feature vectors (zr1 to zrk) extracted for each reference point using bilinear interpolation from Z.
3. **Feature Aggregation**:
   - Cross attention combines information from query point feature (zq) and reference point features (Zr), focusing on relevant parts of the scene based on the query point.
4. **Prediction**:
   - Concatenates aggregated feature vector (z) with query point information (q).
   - Fully connected network with two heads:
     - One head predicts occupancy probability (logits).
     - The other head predicts the flow vector representing object movement.



Evaluation Metrics:

Mean Average Precision (mAP):

- AV2 Dataset: 0.799
- HighwaySim Dataset: 0.510

Soft-IoU:

- AV2 Dataset: 0.480
- HighwaySim Dataset: 0.886

Expected Calibration Error (ECE):

- AV2 Dataset: 0.193
- HighwaySim Dataset: 0.076

Endpoint Error (EPE):

- AV2 Dataset: 0.267
- HighwaySim Dataset: 0.510

Flow Grounded Metrics:

- AV2 Dataset: 0.936
- HighwaySim Dataset: 0.886

Waymo Open Dataset Occupancy and Flow Prediction Challenge Solution: Look Around

Introduction:

Previous methods suffered from the problems of:

1. Anchor-based Goal Predictions
2. Goal-based Multi-Trajectory Prediction

New Approach:

1. Anchor-free Model: The DenseTNT model performs dense goal probability estimation for trajectory prediction, moving away from the traditional anchor-based approach.
2. Dense Goal Probability Estimation: This technique allows for a more detailed and accurate prediction of potential trajectories by considering a dense set of potential goals rather than sparse anchors.

Implementation:

The architecture of DenseTNT is designed to handle dense goal probability estimation efficiently.It includes components for goal scoring, goal selection, and trajectory completion based on the selected goals.

Model Inputs:

The input is prepared for a 512x512 pixel area, with an additional 128 pixels on each side of the target area.

Architecture:

HRNet-based Segmentation Network:

- The core of the model is based on the High-Resolution Network (HRNet) for segmentation tasks.

Spatial Transformer Module:

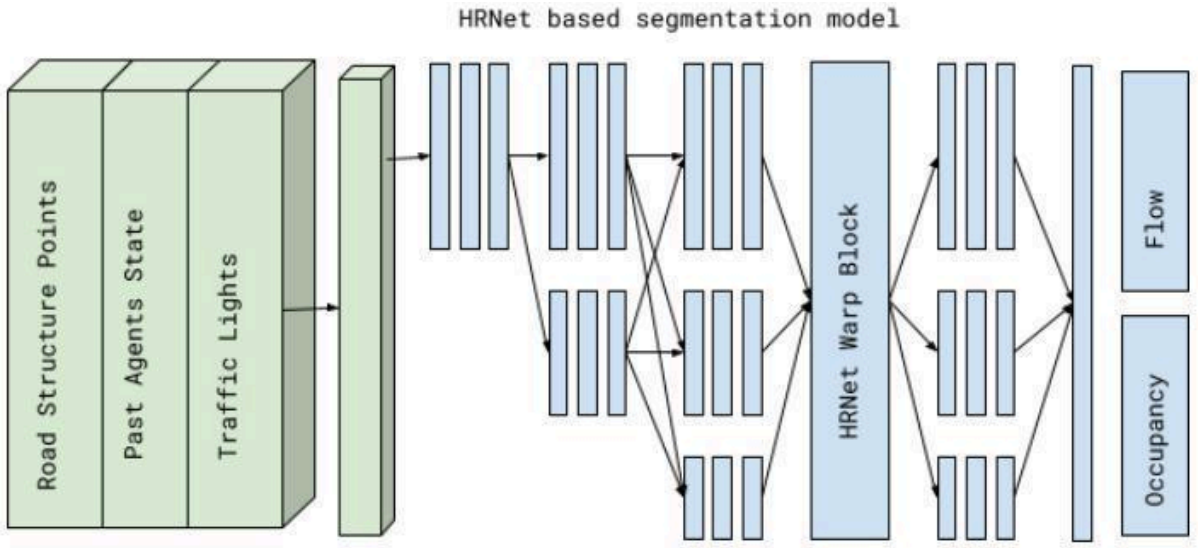
- This module allows the network to spatially transform the input features based on learned parameters, enhancing the model's ability to focus on relevant regions of the input.

Residual Blocks:

- These are used as building blocks within the network to enable the learning of residual mappings. Residual mappings improve the flow of information through the network and enable better feature representation.

Warp Block:

- This block helps to enhance the model's performance by allowing it to predict and project the agents' states on the grid step by step.



<u>Outputs:</u>

- predicts the flow for multiple time steps.
- It also predicts the total occupancy and whether the predicted occupancy is occluded or not.

<u>Results</u>

DenseTNT achieves state-of-the-art performance in trajectory prediction.

It ranked 1st in the Waymo Open Dataset Motion Prediction Challenge.

Evaluation Metrics:

Observed Occupancy:

- AUC (Area Under the Curve): 0.8014
- Soft-IOU (Intersection over Union): 0.2346

Occluded Occupancy:

- AUC (Area Under the Curve): 0.1386
- Soft-IOU (Intersection over Union): 0.0285

Endpoint Error (EPE): 2.6191

Flow Temporal Metrics:

- FT-AUC (Flow Temporal Area Under the Curve): 0.8246
- FT-Soft-IOU (Flow Temporal Soft-IOU): 0.5488

Challenges:

- **Complexity of Dense Predictions**:Handling and processing dense goal probability data increases computational complexity
- **Evaluation and Validation:** Designing appropriate metrics and frameworks to evaluate and validate the model's performance accurately.

VectorFlow: Combining Images and Vectors for Traffic Occupancy and Flow Prediction

Introduction:

Previous methods suffered from the problems of:

1. **Image-based Models**: Utilising convolutional neural networks (CNNs) to analyse traffic images and predict traffic occupancy.
2. **Vector-based Models**: Using vector data such as vehicle positions and trajectories for traffic flow prediction.
3. **Hybrid Models**: Combining image and vector data for enhanced prediction accuracy.

New Approach:

1. **Combination of Images and Vectors**: VectorFlow integrates both image data (e.g., traffic camera footage) and vector data (e.g., vehicle trajectories) to improve traffic occupancy and flow prediction.
2. **Enhanced Prediction Accuracy**: By combining these data types, the model can capture more comprehensive traffic patterns and interactions.

Architecture:

Dual-Input Network: The architecture features two primary inputs: images and vector data.

- **Image Processing Module**: Uses CNNs to extract features from traffic images.
- **Vector Processing Module**: Employs techniques like RNNs or transformer networks to process vector data.

Fusion Layer: Combines the extracted features from both modules for integrated prediction.

Output Layer: Generates predictions for traffic occupancy and flow.

<u>Training and Testing Dataset:</u>

The model described in the paper is trained, evaluated, and tested on the Waymo Open Motion Dataset (WOMD) based on the standard split and the filtered scenarios.

- **Training Details:**
  - **Dataset:** Full training set of WOMD
  - **Batch Size:** 32
  - **Epochs:** 16
  - **Hardware:** 8 Nvidia A10 GPUs

Evaluation Key Performance Indicators (KPIs):

1. **Observed Occupancy:**
   - **AUC (Area Under the Curve):** 0.7548
   - **Soft-IOU (Intersection over Union):** 0.4884
2. **Occluded Occupancy:**
   - **AUC (Area Under the Curve):** 0.1736
   - **Soft-IOU (Intersection over Union):** 0.0448
3. **Improvement in Occluded Metrics:**
   - **Occluded AUC:** Improved by 24.46
   - **Occluded Soft-IOU:** Improved by 47.06
4. **Validation Set Performance:**
   - **AUC:** 0.760
   - **Soft-IOU:** 0.490

Challenges:

- Data Integration: Combining image and vector data effectively without information loss or distortion.
- Computational Complexity: Ensuring the model remains computationally efficient despite processing two types of data.

MotionPerceiver: Real-Time Occupancy Forecasting for Embedded Systems

Introduction:

Previous Methods:

1. **Recurrent Neural Networks (RNNs)**: Typically used for sequence modelling, but can be inefficient for real-time processing.
2. **CNN-based Approaches**: Convolutional Neural Networks (CNNs) used for processing visual data, often resulting in large and fixed-size occupancy images.

Proposed Method:

1. Recursive Latent State Estimation
2. Transformer-based Functions
3. Localised Querying

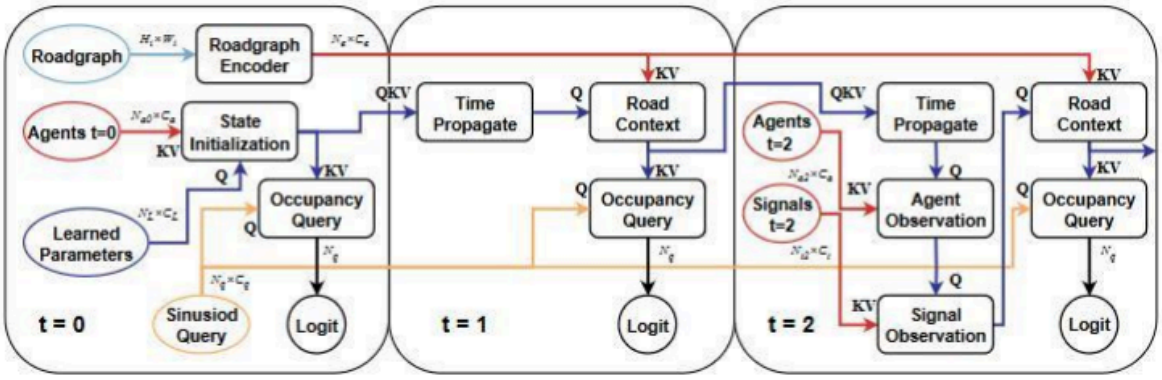
Model Inputs

1. **Agent States**:
    - Positions
    - Orientations
    - Dimensions
    - Velocities
2. **Road Structure Information**:
    - Lane Markings
    - Road Elements

3. **Traffic Light States**:
    - Current Signals

Architecture:

- **Scene Encoding**: The scene is encoded into a latent state using self-attention mechanisms.
- **State Evolution**: The latent state evolves over time through recursive updates.
- **Cross-Attention Mechanisms**: Integrate relevant scene observations dynamically.
- **Localised Querying**: Focuses on points-of-interest to generate occupancy predictions, avoiding the processing of irrelevant regions.



Results:

AUC (Area Under the Curve): 0.365

Soft-IOU (Intersection over Union): 0.196

Challenges

- **Latent State Modelling**: Ensuring the latent state accurately captures the evolving scene.
- **Real-time Processing**: Achieving efficient real-time performance on embedded systems with limited computational resources.

- **Integration of Scene Observations**: Effectively incorporating various scene elements (e.g., traffic signals, road topology) through cross-attention mechanisms.

HOPE: Hierarchical Spatial-temporal Network for Occupancy Flow Prediction

Introduction:

Previous Methods:

1. **Convolutional Neural Networks (CNNs)**: Used for spatial feature extraction in occupancy flow prediction.
2. **Recurrent Neural Networks (RNNs)**: Employed for capturing temporal dependencies in traffic flow data.
3. **Hybrid Approaches**: Combined CNNs and RNNs to handle both spatial and temporal aspects, but often lacked a hierarchical structure to effectively model complex interactions.

Proposed Method:

1. **Hierarchical Modelling**: Introduces a hierarchical network that processes spatial and temporal data at multiple levels of abstraction.
2. **Spatial-temporal Integration**: Seamlessly integrates spatial and temporal features using a unified framework.

Architecture:

1. **Encoders**:
    - **2D CNN Encoder**: Captures spatial features from rasterized bird's-eye view inputs.
    - **Transformer-based Encoder**: Processes sequences to capture temporal dependencies.
    - **Spatial-temporal Sparse 3D CNN Encoder**: Integrates spatial and temporal features.

2. **Aggregator**:
    - **BiFPN Layers**: Facilitate multi-scale feature fusion.
    - **Latent Variable Module**: Adds stochastic modelling capabilities for predicting various possible future scenarios.
3. **Decoder**:
    - **Dilated 3D Convolutions**: Enhance receptive fields for better context understanding.
    - **ConvLSTM Layer**: Refines the predictions by considering temporal correlations.

Results:

- **Performance**: Achieved a Flow-Grounded Occupancy AUC of 0.8389, outperforming other teams in the leaderboard.

Challenges:

- **Model Complexity**: Managing the increased complexity of the hierarchical model while maintaining efficiency.
- **Data Integration**: Effectively integrating spatial and temporal data at multiple hierarchical levels.
- **Scalability**: Ensuring the model scales well with larger datasets and more complex traffic scenarios.

Dataset Information:

We have used Motion dataset from WAYMO.The dataset was framed in a segment of 9 second where 1 history second and rest 8 second future prediction.

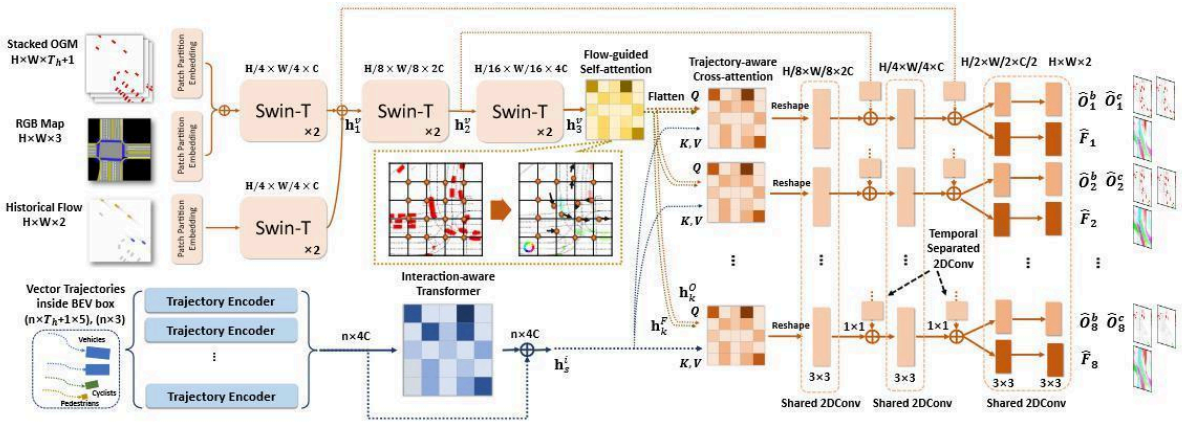
- The dataset comprises 103,354 segments, each lasting 20 seconds. These segments are collected at a frequency of 10Hz, resulting in over 20 million frames for analysis.
- **Sensor Data:** Sensor data from 1 mid-range lidar and 4 short-range lidars is included, offering rich environmental information crucial for autonomous driving research.
- **Object Data:** The dataset contains detailed object data for over 10.8 million objects. Objects are categorised into three classes: Vehicles, Pedestrians, and Cyclists.
- **Map Data:** Map data is available for each segment, covering diverse locations such as San Francisco, Phoenix, Mountain View, Los Angeles, Detroit, and Seattle. This map data includes essential information such as lane centres, boundaries, crosswalks, and more, providing context for the recorded sensor data.

We start of by starting the training of the Efficient Net model we run into a lot of issue of dependencies and dataset size issue as dataset was of about 1.5TB we still somehow manage to get the output as per the problem statement and got the metrics as shown in the figure

```
Metrics:
vehicles_observed_auc: 0.0323185920715332
vehicles_observed_iou: 0.03129800409078598
vehicles_occluded_auc: 0.0029108524322509766
vehicles_occluded_iou: 0.002898802049458027
vehicles_flow_epe: 17.267311096191406
vehicles_flow_warped_occupancy_auc: 0.44335952401161194
vehicles_flow_warped_occupancy_iou: 0.47201868891716003
num_waypoints_with_observed_occupancy: 8
num_waypoints_with_occluded_occupancy: 8
num_waypoints_with_flow: 8
```

The metrics were not too good and also it took us 2-3 days for dataset download and for training also it took us 2 days. We then decided to check with Strajnet model to see if we can achieve more better metrics score.

StrajNet Model



This is the model architecture as we have gone through it above. We start off to set up the dataset and start the preprocessing part with this model where we run into a lot of issues with visualisation of the predicted points of the vehicles occupancy.

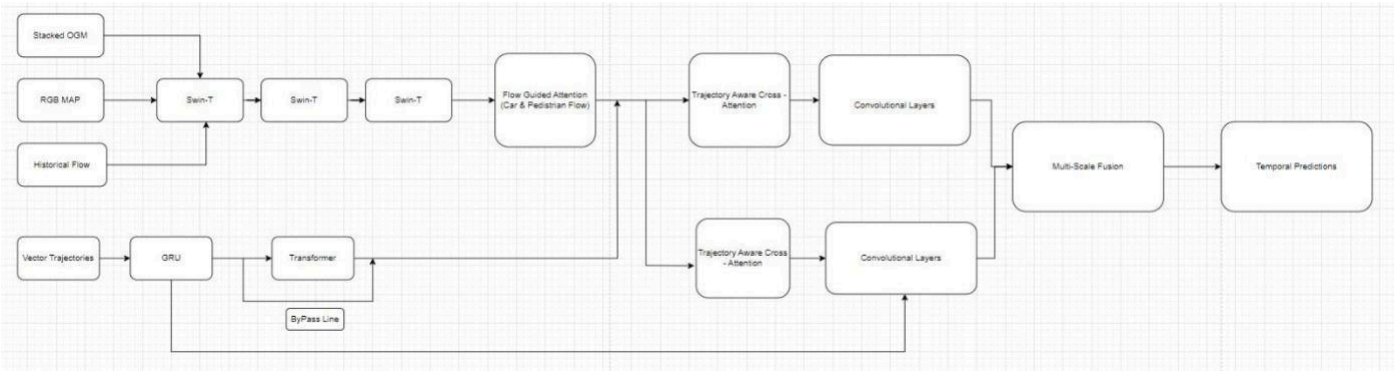
Metrics :

- **loss**: 39.170246
- **val_loss**: 39.290207
- **lr**: 1e-05
- **val_observed_auc**: 0.72664976
- **val_occluded_auc**: 0.09124428 (suspected OCR error)
- **val_observed_iou**: 0.4266094
- **val_occluded_iou**: 0.021474136 (suspected OCR error)
- **val_flow_epe**: 6.0028677
- **val_flow_ogm_auc**: 0.6004355
- **val_flow_ogm_iou**: 0.49730352

We got a proper visualisation with better metrics than the efficient net model so we started to focus on the StajNet Model and look further into what changes can be made to develop a novel solution.
We started to research what novel approach can be brought here to further improve the efficiency of the model to get the relevant output.

As we continue our research we stuck upon how about using LSTM/GRU to remember the past video sequence details as well as that would help model to better get the output of the next sequence as it knows how the environment is changing in the previous scenario and how it is working.

Below is our proposed architecture which is a modification of the strajnet model implemented with GRU.



We selected GRU over LSTM as GRU is a smaller component which makes the training part easier and consumes less power.

Model architecture

swin_tiny_patch4_window7_224

```
 Layer (type)                Output Shape              Param #
=================================================================
 patch_embed (PatchEmbed)    multiple                  17184

 patch_embed (PatchEmbed)    multiple                  3360       (Optional)

 all_norm (LayerNormalizatio multiple                  192        (Optional)

 basic_layer (BasicLayer)    multiple                  1356294    (Optional)

 patch_embed (PatchEmbed)    multiple                  4896       (Optional)

 basic_layer_1 (BasicLayer)  multiple                  1356294

 basic_layer_2 (BasicLayer)  multiple                  1459212

 basic_layer_3 (BasicLayer)  multiple                  3628056

 all_norm (LayerNormalizatio multiple                  192

=================================================================
Total params: 7,825,680
Trainable params: 5,368,080
Non-trainable params: 2,457,600
```

```
 Layer (type)                Output Shape              Param #
=================================================================
 traj_encoder (TrajEncoder)  multiple                  279808

 cross__attention (Cross_Att multiple                  1773312

 layer_normalization_2 (Laye multiple                  768
 rNormalization)

 layer_normalization_3 (Laye multiple                  768
 rNormalization)

 dense_4 (Dense)             multiple                  768

=================================================================
Total params: 2,055,424
Trainable params: 2,055,424
Non-trainable params: 0
```

```
 Layer (type)                Output Shape              Param #
=================================================================
 traj_net (TrajNet)          multiple                  2055424

 cross__attention_t (Cross_A multiple                  425472
 ttentionT)

 cross__attention_t_1 (Cross multiple                  425472
 _AttentionT)

 cross__attention_t_2 (Cross multiple                  425472
 _AttentionT)

 cross__attention_t_3 (Cross multiple                  425472
 _AttentionT)

 cross__attention_t_4 (Cross multiple                  425472
 _AttentionT)

 cross__attention_t_5 (Cross multiple                  425472
 _AttentionT)

 cross__attention_t_6 (Cross multiple                  425472
 _AttentionT)

 cross__attention_t_7 (Cross multiple                  425472
 _AttentionT)

=================================================================
Total params: 5,459,200
Trainable params: 5,459,200
Non-trainable params: 0
```

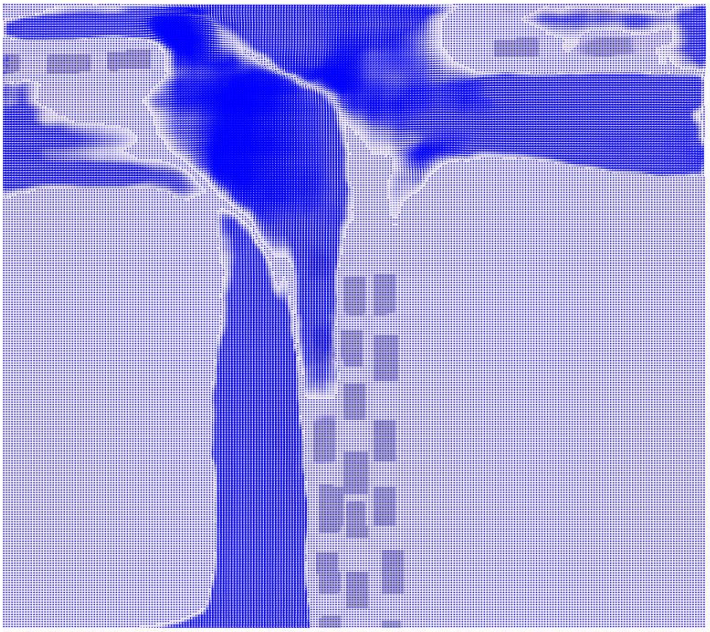
```
 Layer (type)                Output Shape              Param #
=================================================================
 upsample_3 (UpSampling3D)   multiple                  0

 upsample_2 (UpSampling3D)   multiple                  0

 upsample_1 (UpSampling3D)   multiple                  0

 upsample_0 (UpSampling3D)   multiple                  0

 upconv_3_0 (Conv2D)         multiple                  663744

 upconv_2_0 (Conv2D)         multiple                  221312

 upconv_1_0 (Conv2D)         multiple                  110688

 upconv_0_0 (Conv2D)         multiple                  41520

 upsamplef_1 (UpSampling3D)  multiple                  0

 upsamplef_0 (UpSampling3D)  multiple                  0

 upconvf_1_0 (Conv2D)        multiple                  110688

 upconvf_0_0 (Conv2D)        multiple                  41520

 resconv_f (Conv3D)          multiple                  98432

 outconv (Conv2D)            multiple                  866

 resconv_3 (Conv3D)          multiple                  295104

 resconv_2 (Conv3D)          multiple                  98432

 outconv (Conv2D)            multiple                  866

=================================================================
Total params: 1,683,172
Trainable params: 1,683,172
Non-trainable params: 0
```

STrajNet:

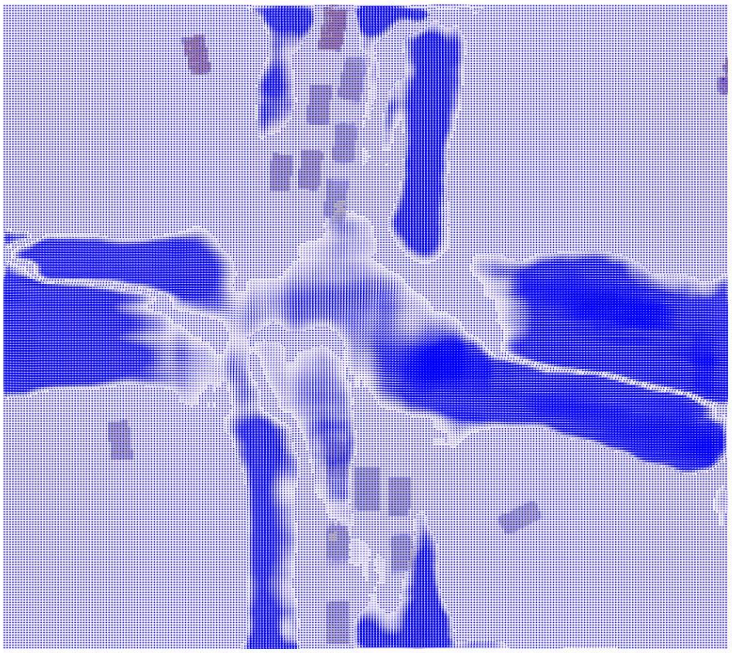
```
 Layer (type)                Output Shape              Param #
=================================================================
 swin_tiny_patch4_window7_22 multiple                  7825680
 4 (SwinTransformerEncoder)

 traj_net_cross_attention (T multiple                  5459200
 rajNetCrossAttention)

 PyrDecoder (Pyramid3DDecode multiple                  1683172
 r)

=================================================================
Total params: 14,968,052
Trainable params: 12,510,452
Non-trainable params: 2,457,600
```

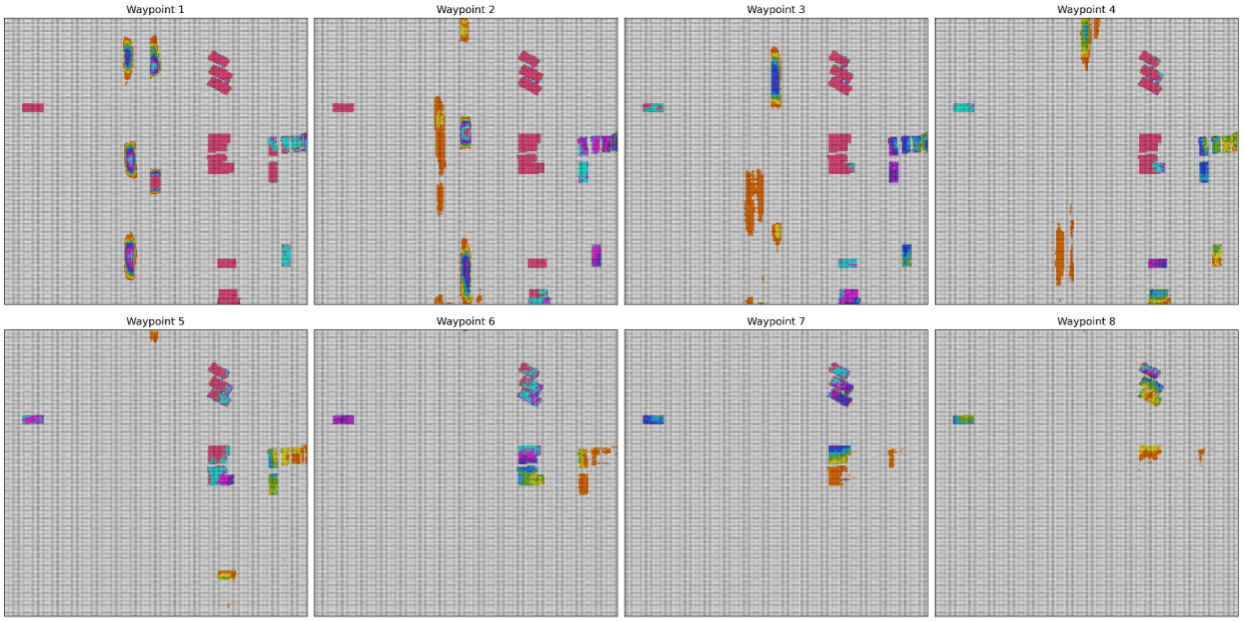
Below are the images of the output that we got after applying the novel approach of the GRU.

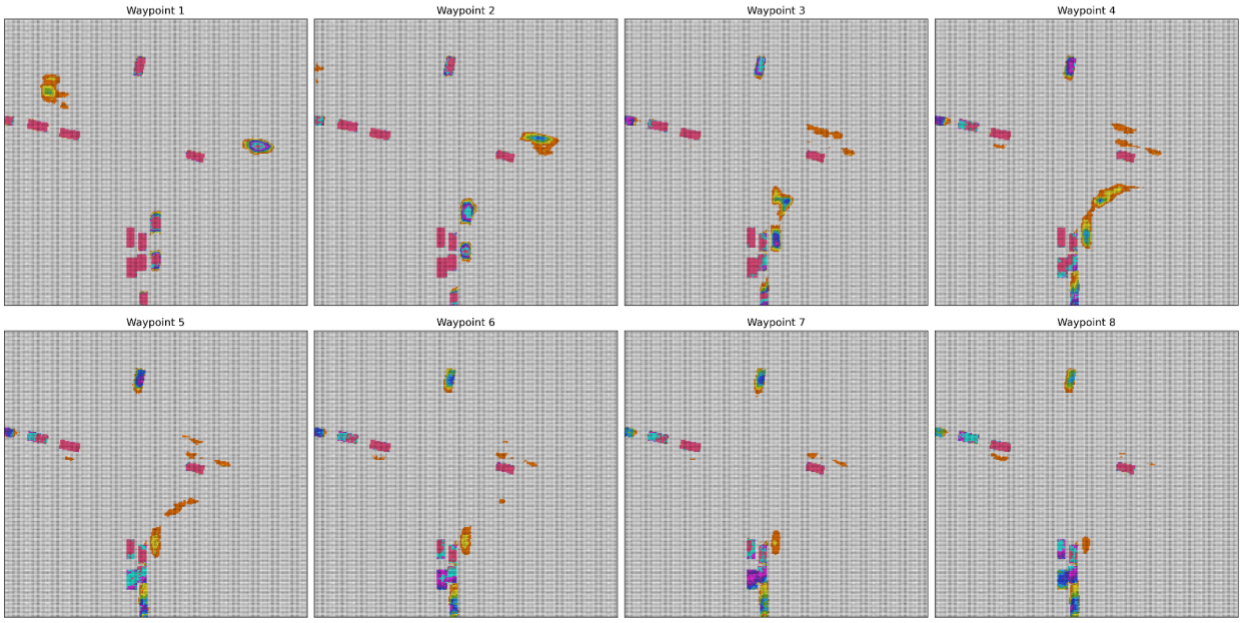
Frame 1/8



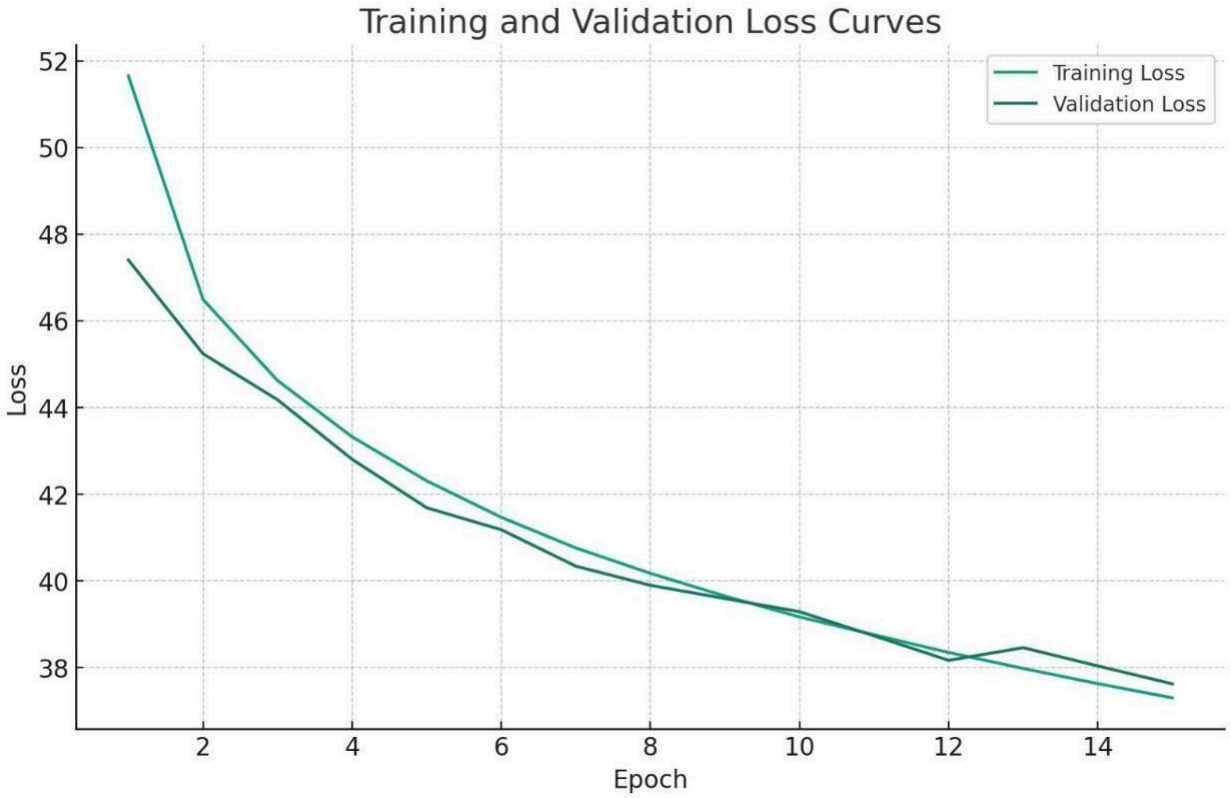
Frame 1/8




Waypoint 1
Waypoint 2
Waypoint 3
Waypoint 4
Waypoint 5
Waypoint 6
Waypoint 7
Waypoint 8


Waypoint 1
Waypoint 2
Waypoint 3
Waypoint 4
Waypoint 5
Waypoint 6
Waypoint 7
Waypoint 8

Below is TRAIN-VAL curve:



Metric	Best Value	Overall Average Value
vehiclesObservedAuc	0.9999959	0.6707341310110121
vehiclesObservedIou	0.93105495	0.394667079980013
vehiclesOccludedAuc	0.0021133367	0.08215408011569333
vehiclesOccludedIou	0.001213575	0.01775754440508122
vehiclesFlowEpe	0.21151373	5.317646466449454
vehiclesFlowWarpedOccupancyAuc	0.9693952	0.5906031259414722
vehiclesFlowWarpedOccupancyIou	0.95420104	0.45259102887026453

```
Model: "traj_net"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 traj_encoder (TrajEncoder)  multiple                  279808

 cross__attention (Cross_Att  multiple                 1773312
 ention)

 layer_normalization_2 (Laye  multiple                 768
 rNormalization)

 layer_normalization_3 (Laye  multiple                 768
 rNormalization)

 dense_4 (Dense)             multiple                  768

=================================================================
Total params: 2,055,424
Trainable params: 2,055,424
Non-trainable params: 0
_________________________________________________________________
Model: "traj_net_cross_attention"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 traj_net (TrajNet)          multiple                  2055424

 cross__attention_t (Cross_A  multiple                 425472
 ttentionT)

 cross__attention_t_1 (Cross  multiple                 425472
 _AttentionT)

 cross__attention_t_2 (Cross  multiple                 425472
 _AttentionT)

 cross__attention_t_3 (Cross  multiple                 425472
 _AttentionT)

 cross__attention_t_4 (Cross  multiple                 425472
 _AttentionT)

 cross__attention_t_5 (Cross  multiple                 425472
 _AttentionT)

 cross__attention_t_6 (Cross  multiple                 425472
 _AttentionT)

 cross__attention_t_7 (Cross  multiple                 425472
 _AttentionT)
```

```
=================================================================
Total params: 5,459,200
Trainable params: 5,459,200
Non-trainable params: 0
_________________________________________________________________
2024-05-10 21:26:17.508638: I tensorflow/compiler/xla/service/service.cc:169] XLA service 0x60958510 initialized for platform CUDA (this does not guarantee that XLA will be used). Devices:
2024-05-10 21:26:17.508672: I tensorflow/compiler/xla/service/service.cc:177]   StreamExecutor device (0): Tesla V100-DGXS-32GB, Compute Capability 7.0
2024-05-10 21:26:17.508685: I tensorflow/compiler/xla/service/service.cc:177]   StreamExecutor device (1): Tesla V100-DGXS-32GB, Compute Capability 7.0
2024-05-10 21:26:17.904617: I ./tensorflow/compiler/jit/device_compiler.h:180] Compiled cluster using XLA!  This line is logged at most once for the lifetime of the process.
Model: "fgmsa"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             multiple                  166272

 layer_normalization_20 (Lay  multiple                 768
 erNormalization)

 conv2d_1 (Conv2D)           multiple                  96

 conv2d_2 (Conv2D)           multiple                  1152

 conv2d_3 (Conv2D)           multiple                  147840

 conv2d_4 (Conv2D)           multiple                  147840

 conv2d_5 (Conv2D)           multiple                  147840

 conv2d_6 (Conv2D)           multiple                  147840

 dropout_52 (Dropout)        multiple                  0

 dropout_53 (Dropout)        multiple                  0

=================================================================
Total params: 767,336
Trainable params: 767,336
Non-trainable params: 0
_________________________________________________________________
Model: "PyrDecoder"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 upsample_3 (UpSampling3D)   multiple                  0

 upsample_2 (UpSampling3D)   multiple                  0

 upsample_1 (UpSampling3D)   multiple                  0

 upsample_0 (UpSampling3D)   multiple                  0
```

Model Architecture Comparison-

Model Architecture Before Changes-

Model	Total Params	Trainable Params	Non-trainable Params
swin_tiny_patch4_window7_224	7,825,680	5,368,080	2,457,600
traj_net	2,055,424	2,055,424	0
traj_net_cross_attention	5,459,200	5,459,200	0
PyrDecoder	1,683,172	1,683,172	0
STrajNet	14,968,052	12,510,452	2,457,600
fgmsa	222,072	222,072	0

Model Architectures After Changes-

Model	Total Params	Trainable Params	Non-trainable Params
swin_tiny_patch4_window7_224	7,825,680	5,368,080	2,457,600
traj_net	2,055,424	2,055,424	0
traj_net_cross_attention	5,459,200	5,459,200	0
PyrDecoder	5,665,348	5,665,348	0
STrajNet	19,717,564	17,259,964	2,457,600
fgmsa	767,336	767,336	0

```
 uplstmconv_0_0 (ConvLSTM2D)  multiple                  3982080

 upconv_2_0 (Conv2D)          multiple                  221312

 upconv_1_0 (Conv2D)          multiple                  110688

 upconv_0_0 (Conv2D)          multiple                  41520

 upsamplef_1 (UpSampling3D)   multiple                  0

 upsamplef_0 (UpSampling3D)   multiple                  0

 upconvf_1_0 (ConvLSTM2D)     multiple                  774528

 upconvf_0_0 (Conv2D)         multiple                  41520

 resconv_f (Conv3D)           multiple                  98432

 outconv (Conv2D)             multiple                  866

 resconv_3 (Conv3D)           multiple                  295104

 resconv_2 (Conv3D)           multiple                  98432

 outconv (Conv2D)             multiple                  866

=================================================================
Total params: 5,665,348
Trainable params: 5,665,348
Non-trainable params: 0
_________________________________________________________________
Model: "STrajNet"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 swin_tiny_patch4_window7_22  multiple                 7825680
 4 (SwinTransformerEncoder)

 traj_net_cross_attention (T  multiple                 5459200
 rajNetCrossAttention)

 fgmsa (FGMSA)               multiple                  767336

 PyrDecoder (Pyramid3DDecode  multiple                 5665348
 r)

=================================================================
Total params: 19,717,564
Trainable params: 17,259,964
Non-trainable params: 2,457,600
```

Summary :

Metric	Before Changes	After Changes
swin_tiny_patch4_window7_224		
- Total Params	7,825,680	7,825,680
- Trainable Params	5,368,080	5,368,080
- Non-trainable Params	2,457,600	2,457,600
traj_net		
- Total Params	2,055,424	2,055,424
- Trainable Params	2,055,424	2,055,424
- Non-trainable Params	0	0
traj_net_cross_attention		
- Total Params	5,459,200	5,459,200
- Trainable Params	5,459,200	5,459,200
- Non-trainable Params	0	0
PyrDecoder		
- Total Params	1,683,172	5,665,348
- Trainable Params	1,683,172	5,665,348
- Non-trainable Params	0	0
STrajNet		
- Total Params	14,968,052	19,717,564
- Trainable Params	12,510,452	17,259,964
- Non-trainable Params	2,457,600	2,457,600
fgmsa		
- Total Params	222,072	767,336
- Trainable Params	222,072	767,336
- Non-trainable Params	0	0

The primary modification of replacing GELU and ELU with the Swish activation function has led to notable changes in the architecture and parameter counts, particularly for the **PyrDecoder**, **STrajNet**, and **fgmsa** models. These modifications has significant increases in the number of parameters, suggest substantial changes in the model structures, aimed at enhancing model performance.