CHALMERS
UNIVERSITY OF TECHNOLOGY

Improving perception systems for autonomous driving

Introducing Mixture of Experts to the PETR Architecture for 3D Object Detection

Master's thesis in Data Science & AI and Computer Systems & Networks

Faton Hoti and Gustav Kalander

DEPARTMENT OF ELECTRICAL ENGINEERING

CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2025
www.chalmers.se

Master's Thesis 2025

Improving perception systems for autonomous driving

Introducing Mixture of Experts to the PETR Architecture for 3D Object Detection

Faton Hoti and Gustav Kalander

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Electrical Engineering
Chalmers University of Technology
Gothenburg, Sweden 2025

Improving perception systems for autonomous driving
Introducing Mixture of Experts to the PETR Architecture for 3D Object Detection
Faton Hoti and Gustav Kalander

Supervisors: Joakim Johnander, Zenseact AB
Niklas Gustafsson, Zenseact AB

Examiner: Henk Wymeersch, Department of Electrical Engineering,
Chalmers University of Technology

Master's Thesis 2025
Department of Electrical Engineering
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Gothenburg, Sweden 2025

Improving perception systems for autonomous driving
Introducing Mixture of Experts to the PETR Architecture for 3D Object Detection
Faton Hoti and Gustav Kalander
Department of Electrical Engineering
Chalmers University of Technology

Abstract

Transformers have become a cornerstone of modern deep learning. Typically, a transformer layer comprises attention, normalization, dropout, and a feed-forward network (FFN). This work investigates the role of the FFN in transformer-based 3D object detection by exploring two modifications: (1) replacing the FFN with a mixture of experts layer to enhance model capacity, and (2) progressively reducing—and ultimately removing—the FFN to assess its necessity. Surprisingly, neither approach led to measurable changes in detection performance, suggesting that the FFN may be functionally redundant in this context. Further experiments revealed that the model retained full performance even when the FFN was entirely eliminated, challenging the conventional assumption that FFNs are indispensable in transformer architectures. These findings raise questions about the necessity of FFNs in perception tasks, contrasting with their established empirically demonstrated importance in NLP. The results also suggest potential avenues for designing leaner, more efficient transformer variants by omitting the FFN.

Keywords: Transformer, 3D Object Detection, Feed-Forward Network, Mixture of Experts, Model Efficiency, Architectural Redundancy

Acknowledgements

We would like to thank our supervisors Joakim Johnander and Niklas Gustafsson for being generous with their time and knowledge throughout this project. They consistently provided thoughtful ideas and suggestions for us to explore. We also thank Henk Wymeersch for being our supervisor and Yuhao Zhang for being our advisor. Finally, thanks to the employees and the other master's students at Zenseact for the ping-pong games and good times!

Faton Hoti and Gustav Kalander, Gothenburg, May 2025

List of Acronyms

Below is the list of acronyms that have been used throughout this thesis listed in alphabetical order:

AD	Autonomous Driving
ADAS	Advanced Driver Assistance Systems
CNN	Convolutional Neural Network
FFN	Feed-forward Network
LiDAR	Light Detection and Ranging
LLM	Large Language Model
mAP	Mean Average Precision
mATE	Mean Average Translation Error
mASE	Mean Average Scale Error
mAOE	Mean Average Orientation Error
MoE	Mixture of Experts
NLP	Natural Language Processing
ReLU	Rectified Linear Unit
GeLU	Gaussian Error Linear Unit
ZOD	Zenseact Open Dataset

Contents

1

Introduction

Deep learning has become essential within many domains, particularly within perception systems in autonomous driving. Since its introduction in 2017 [28], the transformer has emerged as a groundbreaking innovation in deep learning, with widespread adoption in numerous forms [19]. Transformers have enabled significant advancements in various domains such as natural language processing and computer vision. Their potential in perception systems is being studied, as they have demonstrated success in applications like segmentation [31, 6], lane-detection [21], and object detection [5, 32, 22]. Object detection in particular is a critical component of autonomous driving as it allows autonomous systems to recognize and track surrounding objects such as pedestrians, vehicles, and obstacles. Transformers provide a breakthrough alternative to CNNs and RNNs, but their substantial computational demands could restrict use in real-time systems with limited resources [27]. Transformers are computationally demanding for two primary reasons: (1) the quadratic complexity of the self-attention mechanism [28], and (2) the feedforward network (FFN) sublayer, which accounts for a substantial portion of the overall computation of the model [14]. Extensive research has led to highly optimized attention mechanisms, making it a well-addressed problem [15, 30, 1, 16]. However, comparatively less attention has been given to optimizing the FFN sublayer, leaving room for further research in this area.

1.1 Background

One promising approach to mitigating the inefficiency of FFNs is the integration of Mixture-of-Experts (MoE), a strategy first introduced in 1991 by Jacobs et al. [12]. Instead of activating the full FFN, MoEs aim to selectively activate subsets of parameters during inference, reducing overall computation while maintaining model capacity. There are many MoE variants available, some notable ones are Sparsely-Gated MoEs [26], Switch Transformers [10], BASE Layers [17], and σ-MoE [7]. Although MoEs have demonstrated impressive efficiency improvements in large-scale NLP and multimodal models [8, 9, 23, 29], they have seen limited application in vision-based tasks, particularly in 3D object detection. Two honorable mentions are V-MoE proposed by Riquelme et al. [25] and Soft MoE [24], which both use a vision transformer (ViT) with MoE for image classification.

1.2 Theoretical Motivation for Conditional Computation

A key observation in transformer architectures is the sparsity of activations within their FFNs. Empirical studies [18] have shown that a significant portion of neuron activations in FFN layers produce zeroes when subjected to typical input distributions. This phenomenon suggests that large segments of the FFN computation are effectively inactive for any given input, leading to inefficient use of computational resources. The inefficiency arises because traditional FFNs apply a dense transformation — every input is processed by all neurons in the layer, regardless of whether those neurons contribute meaningfully to the output. Given that FFNs often constitute a substantial portion of a transformer's parameters, this dense computation results in considerable redundant processing and memory usage.

If only a subset of parameters is necessary to process a given input, then forcing all parameters to participate in the computation is both wasteful and unnecessary. This observation aligns with the broader concept of conditional computation, where the model dynamically selects which components to activate based on the input, thereby improving efficiency without sacrificing performance. From a theoretical standpoint, the sparsity of activations suggests that the input space is naturally partitioned into regions where different specialized sub-networks would suffice. For example, within the context of an LLM, the specialized sub-networks could correspond to an expert for interpreting verbs, one for punctuation marks, one for adjectives, and so on. Rather than employing a monolithic FFN that attempts to cover all possible scenarios, a more efficient approach would be to route inputs to specialized experts that handle distinct subsets of the data. This is the principle behind MoE layers. MoEs use a gating function that dynamically selects a set number of experts to activate for each input token. The gating mechanism in MoEs is typically trained alongside the experts themselves, learning to distribute inputs across the experts in a mostly uniform way. Crucially, this approach preserves the expressive power of the original FFN since all experts are available when needed while avoiding the inefficiency of dense computation.

We show in Chapter 5 that this phenomenon is indeed apparent in practice for our model. The observed sparsity in FFN activations indicates that much of its computation is redundant. By adopting MoEs, we try to maintain model accuracy while reducing the computational overhead. Additionally, MoEs provide an efficient way to expand model capacity without a corresponding increase in computational cost. Unlike traditional FFNs, where adding more neurons directly increases the cost of every forward pass, MoEs allow for larger pools of experts while keeping the per-input computational budget constant by limiting the number of active experts.

1.3 Goals

This thesis aims to explore how MoEs can enhance Transformer-based 3D object detection models. The primary objective of this thesis is to integrate a specific

MoE variant as a replacement for the traditional FFN sublayer of a state-of-the-art Transformer-based model. The concrete goals of the project consist of (1) adapting a state-of-the-art 3D object detector to the Zenseact Open Dataset (ZOD); (2) replacing the FFN sublayer with a MoE layer; (3) evaluating the model with respect to training time, inference time, and predictive performance.

1.4 Limitations

Only a single state-of-the-art 3D object detection model will be used in this study, without comparison to alternative models. While various MoEs layer variants exist, only one specific variant will be considered. The study will not focus on designing custom MoE layers or comparing different implementations.

2

Theory

This chapter provides the technical foundation required to understand the methods, models, and results presented in the remainder of the report. It introduces the core concepts of neural networks, transformer architectures, attention mechanisms, and the mixture-of-experts (MoE) paradigm.

2.1 Neural Networks

An artificial neural network takes a vector of numbers as inputs, performs some computation and produces a vector of numbers as outputs. Neural networks are function approximators, so examples of inputs and desired outputs (known as training data), can be used to *train* the neural network to approximate the desired outputs.

The basic building block of an artificial neural network is the neuron. A neuron typically has a set of trainable parameters $b, w_1, w_2, \ldots, w_n$, takes a number of inputs $x_1, x_2, \ldots, x_n$ and produces an output

$$y = f\left(b + \sum_{i=1}^{n} w_i x_i\right) \tag{2.1}$$

where f is a non-linear *activation function*, e.g., Rectified Linear Unit (ReLU).

A basic deep neural network consists of some input neurons, some hidden neurons and some output neurons. The neurons form a directed acyclic graph, where the hidden and output neurons use the values of earlier neurons as inputs. The values of the input neurons are set to the neural network's input vector. Activations are then propagated through the neural network according to Equation (2.1) to the output neurons.

A common structure known as fully connected neural networks organizes the neurons in layers: an input layer, a number of hidden layers and an output layer. Each neuron in one layer is connected to every neuron in the next layer as shown in Figure 2.1. This structure allows an entire layer to be efficiently computed using a matrix multiplication. If the input to a layer is a column vector $\mathbf{x} \in \mathbb{R}^n$, the weights of the layer are stored in a matrix $W \in \mathbb{R}^{m \times n}$, and the biases are stored in a vector $\mathbf{b} \in \mathbb{R}^m$, then the output of the layer is given by

$$\mathbf{y} = f(W\mathbf{x} + \mathbf{b})$$

where the activation function f is applied element-wise.

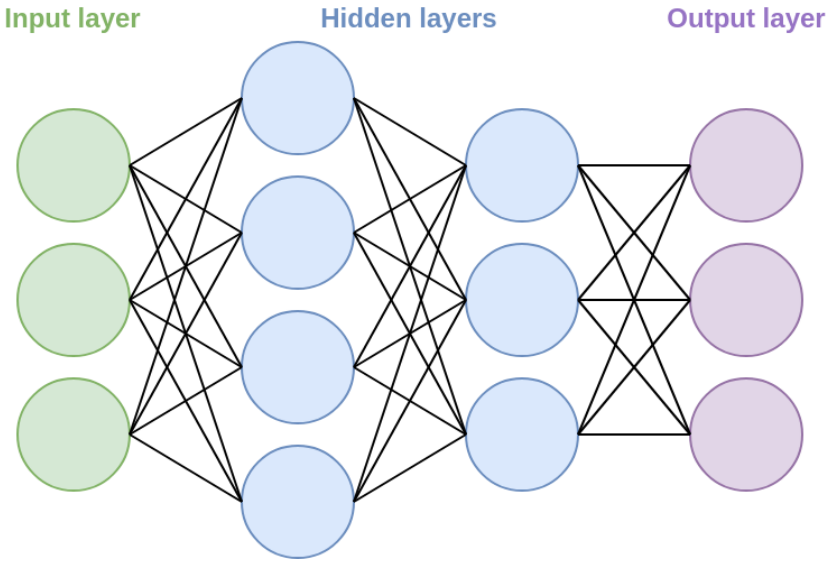


Figure 2.1: An example of a small fully connected neural network with three input neurons, two hidden layers with four and three neurons respectively, and three output neurons.

2.1.1 Backpropagation

The neural network parameters are optimized with the backpropagation algorithm. In supervised learning, where there are labeled examples, the network's output is compared to the ground truth labels and a loss value is calculated. The loss value measures how far from the correct answers the predictions were, so the goal of backpropagation is to minimize the loss. The neural network and the loss function are designed to be differentiable, so this can be done by calculating the partial derivative of the loss with respect to each parameter in the neural network, and then updating the parameters in the direction of greatest loss reduction. Backpropagation calculates these partial derivatives efficiently with the chain rule in a single backward pass through the neural network.

2.2 Transformer

Before transformers, sequence modeling was done via Recurrent Neural Networks (RNNs) and their variants such as Gated Recurrent Unit (GRU) and Long Short-Term Memory (LSTM) networks. These are autoregressive models that process input sequentially, making them computationally inefficient for long sequences and limiting their ability to capture long-range dependencies. The Transformer architecture addressed these limitations by replacing recurrence with attention, allowing parallel processing and improved contextual learning [28].

A transformer is a deep learning architecture that processes input sequences in parallel, relying on the attention mechanism instead of recurrent structures. The core

components include attention and a feed-forward network. The attention algorithm is the fundamental part of a transformer. For each token, the mechanism calculates an attention score for every other token in the sequence, including itself, yielding a similarity measure between the tokens. By allowing each token to attend to all other tokens, the mechanism enables the capturing of contextual dependencies across the entire sequence. This is one of the major improvements over traditional models like RNNs, which struggle with long-range dependencies.

2.2.1 Attention

Let $W_Q \in \mathbb{R}^{d_{\text{in}} \times d_k}$, $W_K \in \mathbb{R}^{d_{\text{in}} \times d_k}$ and $W_V \in \mathbb{R}^{d_{\text{in}} \times d_v}$ be the trainable weight matrices for the query, key, and value projections, respectively. If $X \in \mathbb{R}^{n \times d_{\text{in}}}$ is the input sequence matrix, then $Q = XW_Q \in \mathbb{R}^{n \times d_k}$, $K = XW_K \in \mathbb{R}^{n \times d_k}$, and $V = XW_V \in \mathbb{R}^{n \times d_v}$ are the projected query, key, and value matrices. Here, d_{in} is the input feature dimension, d_v is the value dimension and d_k is the key dimension. Typically, d_v is set to be equal to d_k.

The attention scores are computed using the scaled dot-product attention formula:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V \tag{2.2}$$

2.2.2 Cross-Attention

Compared to attention, cross-attention executes the attention algorithm on two different data sequences. For example, in 3D object detection, cross-attention can be performed to allow learned query embeddings, representing e.g., potential objects, to attend to image features extracted from a 2D backbone. The result is the fusing of information from the image features into the queries, allowing for the refinement of their representation. Thus, cross-attention can be seen as a tool that facilitates information exchange across modalities. Cross-attention can be expressed as

$$\text{CrossAttention}(Q_1, K_2, V_2) = \text{softmax}\left(\frac{Q_1 K_2^T}{\sqrt{d_k}}\right)V_2. \tag{2.3}$$

The subscripts refer to which input the respective quantities come from.

2.2.3 Multi-Head Attention

Another version of attention is multi-head attention, where instead of using a single attention mechanism, h attention heads operate in parallel. The heads have identical

structure, but different parameters. For head i, the projections are computed as:

$$Q_i = XW_Q^i \in \mathbb{R}^{n \times \frac{d_k}{h}}$$
$$K_i = XW_K^i \in \mathbb{R}^{n \times \frac{d_k}{h}}$$
$$V_i = XW_V^i \in \mathbb{R}^{n \times \frac{d_v}{h}}$$

where $W_Q^i, W_K^i \in \mathbb{R}^{d_{\text{in}} \times \frac{d_k}{h}}$ and $W_V^i \in \mathbb{R}^{d_{\text{in}} \times \frac{d_v}{h}}$ are the per-head projection matrices. The outputs of all heads are concatenated and projected back to the original dimension:

$$\text{MultiHead}(Q, K, V) = \text{Concat}(z_1, ..., z_h)W^O \tag{2.4}$$

where $z_i = \text{Attention}(Q_i, K_i, V_i)$ and the output projection matrix $W^O \in \mathbb{R}^{d_v \times d_{\text{out}}}$ (typically $d_{\text{out}} = d_{\text{in}}$) combines the heads' outputs.

2.2.4 Positional Embedding

One direct effect of processing sequences in parallel rather than sequentially is that transformers lack the ability to understand the order or spatial arrangement of input tokens. This is problematic within contexts where such context is crucial. To address this, positional embeddings are created to inject the missing information about the structure of the input into the model. In the context of 3D object detection, positional embeddings are typically derived from 3D spatial positions, e.g., 3D coordinates, Additionally, complementary image features can also be incorporated to enhance the representation. These embeddings introduce spatial context and enables the attention algorithm to reason about the relative positions of objects in 3D space.

2.3 Mixture of Experts

Mixtures of Experts (MoE) is an architecture designed to improve computational efficiency and scalability by dynamically selecting a subset of specialized networks called experts for each input token. The two core components of a MoE layer are the expert networks and the gating function. Each expert function is in and of itself a smaller FFN that specializes in different types of input patterns. A gating network is used to decide which experts should be activated for a given token.

2.3.1 Gating Function

Gating functions, also known as routing functions, can be designed in many different ways. Here follows a brief overview of some common types of gating functions. **Dense** gating functions are used for maximizing performance and work by activating all experts. It has high computational demand as it uses all experts on each forward pass, but is easy to use. **Sparse** gating functions activate only a subset of

experts and offer a lower computational demand compared to activating all experts. Sparse gating functions typically require load balancing to avoid expert collapse, where some experts are used almost always while others are redundant. Load balancing is usually achieved either by adding random noise to expert selections or by using an auxiliary load balancing loss to incentivize an even usage of experts. Sparse gating functions come in several different flavors; (1) **Token-Choice**, which assigns several experts per input token and requires training. An expert capacity can be added to introduce a threshold for the number of tokens that an expert can process. (2) **Non-trainable Token-Choice**, is similar to Token-Choice, but replaces the gating functions that require dynamic training with static ones. An example of such a function is to use a hash which assigns experts tokens based on the hash of the token. The idea is to mitigate the need for additional gating network parameters. (3) **Expert-Choice**, switches the responsibility from the gating function to the individual experts. Each expert themselves choose e.g., top-k tokens they will process. This circumvents the necessity for auxiliary load balancing losses during training and ensures a uniform distribution of tokens across experts. However, this can lead to uneven token coverage, where some tokens are processed by multiple experts while others may be ignored entirely.

2.3.2 Hyperparameters

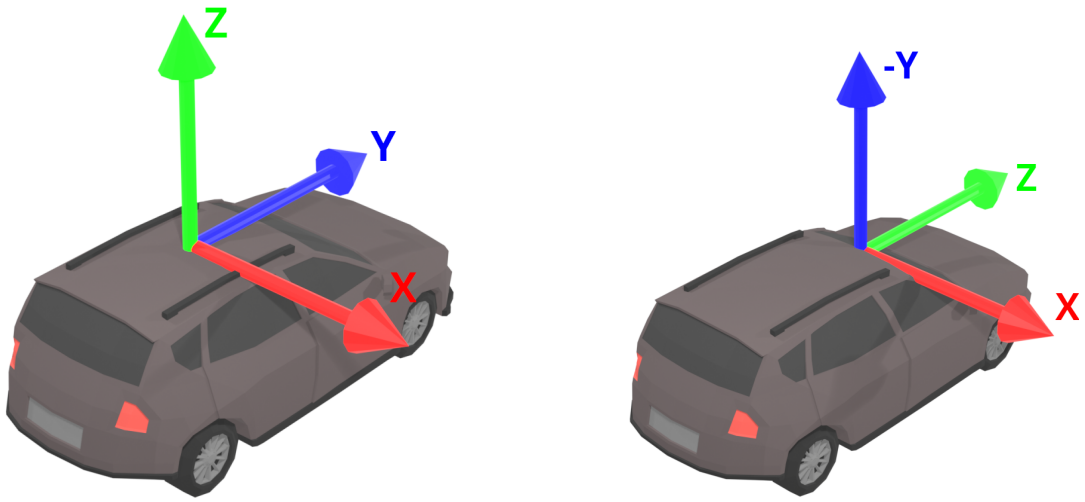
Table 2 in the survey of MoE design choices by Cai et al. [4] provides a comparative overview of MoE configurations across various models; highlighting how different architectures adjust key hyperparameters to balance predictive performance and computational efficiency. Key hyperparameters include the **number of experts per MoE layer**, where recommendations suggest a maximum of 64 experts, though configurations as few as 8 or 16 are also effective depending on the use case. The **number of activated experts within a layer**. The **size of each expert** can vary widely — from 1024 to 8192, and even up to 16k or 32k dimensions — affecting both capacity and resource demands. Another important factor is the **placement frequency of MoE layers**. Strategies range from alternating every other layer (1/2), every fourth layer (1/4), or even applying MoE layers everywhere (1/1). Lastly, the **choice of activation function** plays a role, with a trend shifting from traditional ReLU to functions like GeLU, GeGLU, and SwiGLU.

2.4 Coordinate Frame Transformations

Within the context of collecting data for use in e.g. 3D object detection, the data often come from multiple sensors, such as cameras and LiDAR, each operating in its own coordinate system. It is common to have to do transformations between these coordinate frames. These transformations rely on linear algebra principles, particularly matrix operations, to map points from one coordinate system to another. This section outlines the key steps and concepts involved in these transformations.

2.4.1 Coordinate Systems

The LiDAR system is defined with its origin at the mount point on the vehicle roof, as illustrated in Figure 2.2a. In this coordinate frame, the y-axis points forward, the x-axis extends to the right, and the z-axis points upward. For the camera system (Figure 2.2b), the origin is at the camera itself, with the z-axis oriented forward, the x-axis pointing to the right, and the y-axis directed downward. All coordinates are expressed in meters.



(a) LiDAR coordinate system **(b)** Camera coordinate system

Figure 2.2: Coordinate systems for (a) LiDAR and (b) camera.

2.4.2 Intrinsics

Intrinsic parameters describe the internal characteristics of a sensor, such as focal length and optical center for cameras. These parameters are represented as a matrix K and are used to transform points from the sensor's coordinate system to the image plane. For a camera, the intrinsic matrix K is defined as:

$$K = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix},$$

where f_x and f_y are the focal lengths, and (c_x, c_y) is the optical center.

2.4.3 Extrinsics

Extrinsic parameters describe the relative position and orientation between two sensors, defining how one is spatially aligned with respect to the other. These are represented as a rotation matrix R and a translation vector t. Together, they form a 4x4 transformation matrix T in homogeneous coordinates, which maps points from one sensor's coordinate system to another. The transformation matrix T is defined as

$$T = \begin{bmatrix} R & t \\ 0 & 1 \end{bmatrix} \ ,$$

where R is the 3x3 rotation matrix and t is the 3x1 translation vector. Note that due to the nature of affine geometry, homogenous coordinates are necessary in order to allow baking in translation into the matrix so to allow for computational efficiency when composing transformations.

2.4.4 Kannala-Brandt Projection and Unprojection

The Kannala-Brandt projection model is a non-linear barrel distortion projection model that yields wide-angle lenses. Unlike the pinhole camera model, the Kannala-Brandt model does not allow for a straightforward matrix transformation K nor inversion using K^{-1} due to its non-linear nature. Projecting is done by first converting 3D camera coordinates to polar coordinates, then applying a polynomial distortion function to the radial angle. Unprojection involves solving a non-linear equation to map points from the image plane back to camera space. For more detailed information on projection and unprojection, check the original paper [13].

2.4.5 Camera Space to LiDAR Space

Transforming points from camera space to LiDAR space involves applying the 4x4 transformation matrix T. Given a point $(x_c, y_c, z_c, 1)$ in camera space (in homogeneous coordinates), its corresponding point in LiDAR space $(x_l, y_l, z_l, 1)$ is computed as:

$$\begin{bmatrix} x_l \\ y_l \\ z_l \\ 1 \end{bmatrix} = T \begin{bmatrix} x_c \\ y_c \\ z_c \\ 1 \end{bmatrix},$$

where T is the transformation matrix for going from camera space to LiDAR space.

2.5 PETR

The PETR (Position Embedding Transformation) model for 3D object detection is based on the DETR [5] framework and modifies it with the novel idea of transforming 2D features from the backbone to 3D-aware features. This allows the object queries to be updated under an environment where spatial context has been introduced as they now undergo cross-attention directly with 3D position-aware features, rather than 2D features. The main components of the PETR architecture are: a Convolutional Neural Network (CNN) that produces 2D image features; a 3D position-encoder which produces 3D-position-aware features; a traditional transformer-decoder; a classification head and a regression head that produce

a set of classifications and box predictions. The box predictions undergo Hungarian matching with the annotations followed by loss calculation. An overview of the model architecture can be seen in Figure 2.3.

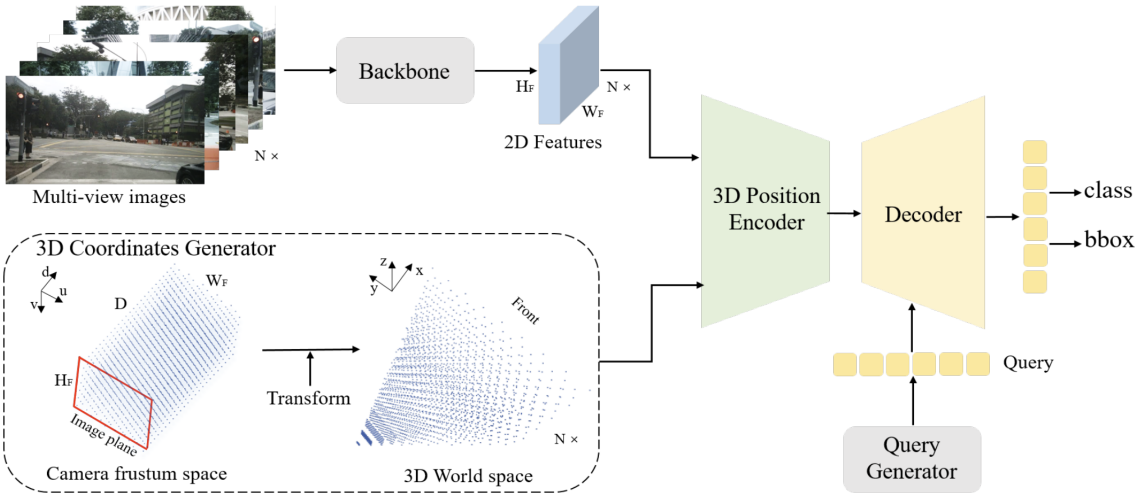


Figure 2.3: Overview of the PETR architecture from the paper [22]. The model merges 2D image features with generated 3D coordinates, creating a positional embedding that encodes spatial awareness in three dimensions. The new positional embedding along with object queries go through cross-attention inside the transformer. The output of the transformer goes through a classification head and a regression head. Figure reproduced from Liu et al. [22] with permission.

2.5.1 Backbone

The backbone of PETR is the ResNet50 CNN [11], which extracts 2D image features from input camera views. These features capture visual patterns, but lack explicit 3D spatial information.

2.5.2 3D Coordinates Generator

The 3D coordinates generator produces spatial-aware embeddings for the transformer through a multi-step process. First, it creates a 3D mesh grid in image coordinates, with height and width dimensions normalized to the padded image size. For depth, it generates discretized bins between a minimum depth and maximum detection range. These image coordinates are then unprojected into 3D camera space. This effectively means that each image pixel defines a 3D ray originating from the camera center through that pixel, and 3D coordinates are sampled at evenly spaced depth intervals along this ray. The points are subsequently transformed into LiDAR coordinates using the camera-to-LiDAR extrinsic matrix. The resulting 3D coordinates are normalized to $[0, 1]$ range based on predefined position bounds. Finally, coordinates outside the valid range are masked out, and the final 3D positional embedding consists of fourier features of these normalized coordinates.

2.5.3 Transformer Decoder

The transformer decoder refines a set of learnable object queries through iterative cross-attention with the 3D-aware image features. Each decoder layer enables the queries to attend to different parts of the 3D scene by processing the enriched positional embeddings generated earlier. Unlike DETR3D, where queries interact with 2D features, PETR's decoder leverages the enriched positional embeddings, yielding potential for a better spatial understanding.

2.5.4 Classification Head

The classification head predicts class probabilities using a focal loss [20]

$$\mathcal{L}_{\text{cls}} = -\alpha(1 - p_t)^{\gamma} \log(p_t) \tag{2.5}$$

where p_t is the estimated probability for the target class, γ is a focusing parameter, and α is a balancing hyperparameter for class frequencies.

2.5.5 Regression Head

The regression head estimates 3D bounding box parameters $b = (x, y, z, w, l, h, \theta) \in \mathbb{R}^7$ using an L1 loss

$$\mathcal{L}_{\text{reg}} = \lambda \cdot \|b_{\text{pred}} - b_{\text{gt}}\|_1 \tag{2.6}$$

where λ controls the relative weight of the regression loss.

2.6 Evaluation Metrics

The development of a machine learning model requires the ability to evaluate its predictive performance. This is typically done by evaluating the model performance on unseen so-called test data with respect to different metrics. Depending on the task at hand, numerous evaluation metrics can be used to infer insights about the effectiveness of the model.

2.6.1 mean Average Precision (mAP)

The Average Precision (AP) metric evaluates object detection performance by assessing the precision-recall trade-off across confidence thresholds. Predictions are sorted by confidence, and each is matched to a ground truth annotation if their 3D center points lie within a specified distance threshold, e.g., 2 meters. A true positive (TP) is recorded if a match is found, and the matched annotation is removed to prevent reuse. This process yields a ranked list of predictions classified as TP or false positive (FP). A precision-recall curve is then constructed by varying the confidence threshold, and AP is computed as the area under this curve. The mean Average

Precision (mAP) extends this by averaging AP scores across multiple object classes and distance thresholds.

2.6.2 mean Average [Translation/Scale/Orientation] error

The nuScenes detection metrics provide a framework for evaluating 3D object detection performance by examining both the quantity and quality of true positive (TP) predictions. For each correctly detected object (TP), we calculate how precisely the prediction aligns with the ground truth across several geometric dimensions. These component metrics, described in detail in Caesar et al. [3], are then aggregated into the following mean average errors. The translation accuracy is measured through **mATE** (mean Average Translation Error), which computes the Euclidean distance between predicted and ground-truth centers in the 2D plane. For scale assessment, **mASE** (mean Average Scale Error) evaluates dimensional accuracy by first aligning the prediction's center and orientation with the ground truth, then calculating 1 - IOU (Intersection over Union). Orientation is captured by **mAOE** (mean Average Orientation Error), defined as the smallest yaw angle difference between prediction and ground truth in radians. The **NDS** (nuScenes Detection Score) serves as the primary composite metric, combining all error components in the form of a weighted sum. While nuScenes additionally includes **mAVE** (mean Average Velocity Error) for velocity estimation, this metric is not applicable to the ZOD and is not used.

3

Data

Deep learning requires training on large datasets that provide an extensive representation of the problem domain. To ensure high quality training it is important for the dataset to provide a rich diversity of scenarios and environments, consist of relevant data, and to include accurate annotations. For example, within the context of perception systems, a frame containing heavy motion blur, overexposure, or an obstructed view can introduce unwanted noise.

3.1 Zenseact Open Dataset

The Zenseact Open Dataset (ZOD) is a large-scale and diverse multimodal dataset designed for use in autonomous driving research [2]. It includes high-resolution data with accompanying annotations. ZOD includes roughly 100,000 annotated single-frame camera images, RADAR sensor data, LiDAR sensor data, and sequence recordings. The data is collected across various geographic locations, weather conditions, and driving scenarios; making it a suitable resource for training. The annotations include 2D and 3D bounding boxes, semantic segmentation, and detailed metadata. The dataset consists of

- **Single Frames**: Contains individual frames with synchronized sensor data and annotations.
- **Sequences**: Includes continuous sequences of frames, allowing for temporal analysis and tracking.
- **Drives**: 29 multi-minute sequences, with full sensor coverage.

We will only be using the single frame data. ZOD also provides calibration data for all sensors, ensuring accurate sensor fusion and alignment.

3.2 Data Preprocessing

Due to the availability of a comprehensive Software Development Kit (SDK) provided with the Zenseact Open Dataset (ZOD), minimal manual preprocessing was required. The SDK facilitates efficient data extraction, synchronization, and annotation handling. We used the `mmdetection3d` framework to train our model, with

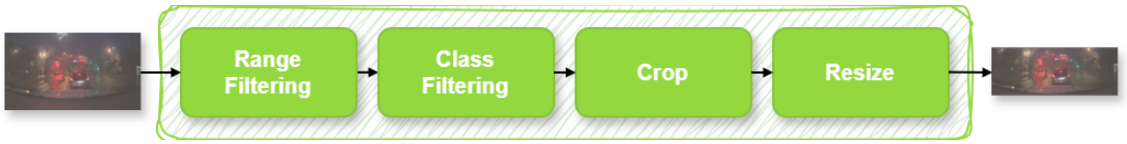


Figure 3.1: Overview of the image transformation pipeline applied to each input frame before training. This includes range filtering, class filtering, cropping, and resizing.

primary pre-processing steps focused on formatting the data to align with its requirements. Each input image undergoes a series of transformations as depicted in Figure 3.1.

3.2.1 Range Filtering

The filter removes bounding boxes that fall outside a predefined range, ensuring the model processes only relevant objects near the ego vehicle. This is done for primarily two reasons. Objects close to the ego vehicle are more important than objects hundreds of meters away. Annotations are typically done from the LiDAR, which are reliable only within a range. Beyond that range it is difficult to create good annotations.

The chosen valid detection area spans from 0 to +100 meters forward, laterally between −50 to +50 meters, and vertically between −3 and +5 meters. To justify this range selection, we analyze the spatial distribution of ground truth 3D bounding boxes in the dataset. Figure 3.2 shows a scatter plot of bounding box centers projected onto the XY plane, which is the ground plane relative to the ego vehicle. The plot reveals that the majority of objects lie within the selected lateral and longitudinal bounds, with progressively sparse detections beyond these limits. Additionally, Figure 3.2 also presents a histogram of bounding box distances along the LiDAR coordinate system Y-axis, demonstrating that most objects are concentrated within 100 meters ahead of the ego vehicle. Beyond this range, the frequency of objects drops significantly, and their small size in the image space makes reliable detection challenging. Thus, the chosen range filtering parameters effectively balance computational efficiency with coverage of the most relevant detection regions.

3.2.2 Class Filtering

The Zenseact Open Dataset (ZOD) provides a rich hierarchy of object classes, including detailed subcategories under broader top-level classes. For this study, we focus on three primary top-level classes: `Vehicle`, `VulnerableVehicle`, and `Pedestrian`. They are the most common classes in the dataset. Figure 3.3 illustrates the distribution of the top-10 sub-classes across the dataset. The `Vehicle` class dominates the distribution, followed by `VulnerableVehicle` (e.g., bicycles, motorcycles) and `Pedestrian`. Further details on the class hierarchy and annotation methodology can be found in Alibeigi et al. [2].

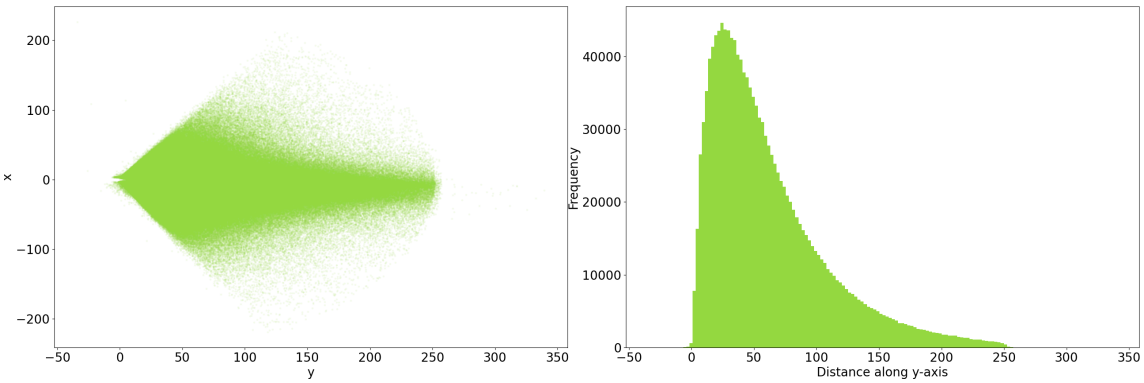


Figure 3.2: Left: XY scatter plot of 3D bounding box centers, projected onto the ground plane relative to the ego vehicle. Most objects are concentrated within the lateral bounds of $[-50, 50]$ meters and forward range of up to 100 meters. Right: Histogram of object counts along the forward Y-axis, illustrating a sharp drop in frequency beyond 100 meters.

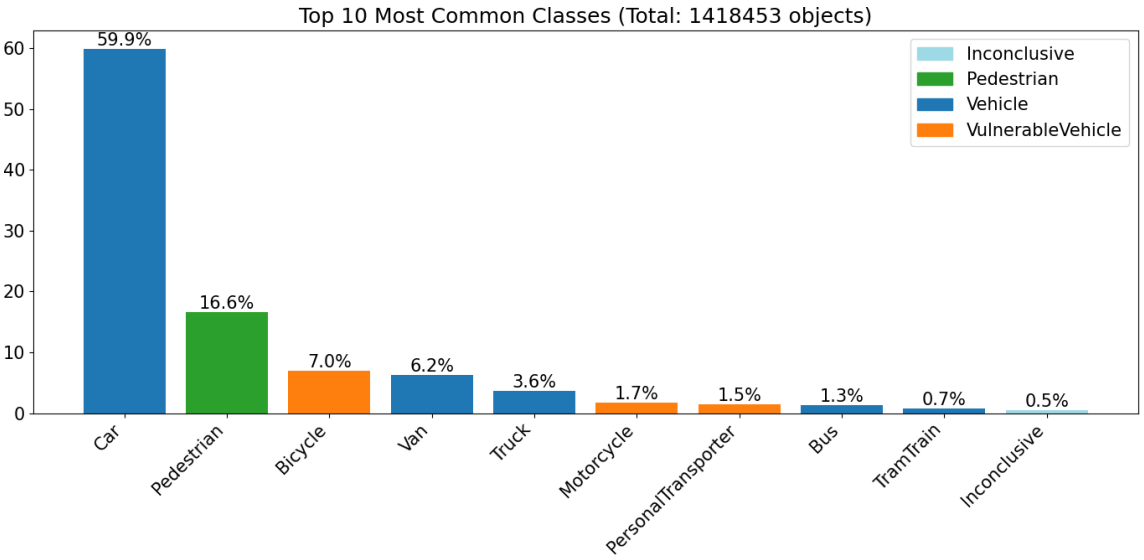


Figure 3.3: Distribution of the 10 most frequent annotated object sub-classes in the ZOD.

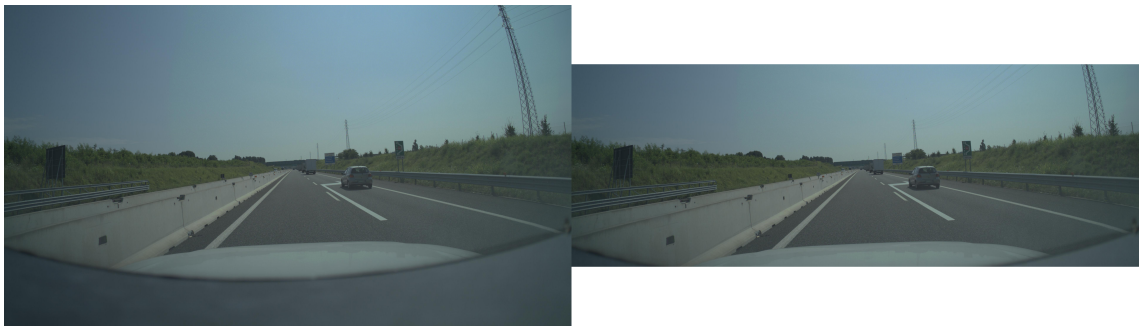


Figure 3.4: Effect of cropping N pixels from the top and bottom: Original image (left) versus the cropped version (right), which excludes non-informative regions such as the car hood and sky.

3.2.3 Crop

After visually evaluating different values of N to ensure the retention of important regions across the dataset, we settled on $N = 400$. It is large enough to improve computational efficiency while not overly aggressive in cropping to preserve valuable image content. As seen in Figure 3.4, we remove N rows of pixels from both the top and bottom of the image, eliminating non-informative regions (car hood and sky). The crop operation translates the optical center c_y in the intrinsic matrix by $-N$ pixels to account for the removed upper portion. Note that the translation is not $-2N$ as cropping from the bottom edge has no effect on the optical center.

3.2.4 Resize

The input images are resized from their original resolution ($W_{orig} \times H_{orig}$) to a target size ($W_{target} \times H_{target}$) while maintaining aspect ratio. Let $s_x = W_{target}/W_{orig}$ and $s_y = H_{target}/H_{orig}$ be the width and height scaling factors respectively. The resulting updated intrinsic matrix becomes

$$K_{resized} = \begin{bmatrix} s_x f_x & 0 & s_x c_x \\ 0 & s_y f_y & s_y c_y \\ 0 & 0 & 1 \end{bmatrix},$$

where the focal lengths and optical center coordinates are scaled proportionally to maintain correct geometric relationships. When aspect ratio preservation leads to unequal scaling ($s_x \neq s_y$), we use the smaller scaling factor for both dimensions to prevent distortion, adjusting the final image dimensions accordingly.

4

Modified PETR Architecture

In the process of adapting PETR, various modifications and additions to the model are implemented. The following chapter will walk through the architecture of the original model as well as the additional components and changes introduced.

4.1 Positional Embeddings

The original architecture was designed with the expectation that the input images have no camera distortion. However, the images in ZOD were captured using a lens following the Kannala-Brandt projection model. To accommodate this difference, the generation of positional encoding 3D coordinates had to be reworked. As seen in Figure 2.3, the original architecture transforms a meshgrid into the shape of a traditional camera frustum. The reworked version, shown in Figure 4.1, produces a distorted frustum shape that accounts for the lens distortion by incorporating the Kannala-Brandt unprojection coefficients during coordinate generation. This procedure pre-distorts the meshgrid to match the distortion characteristics of the camera.

4.2 Swapping In a MoE layer

The original PETR model uses a traditional transformer in which a standard FFN is placed within each of its transformer decoder blocks. In this thesis, we experiment

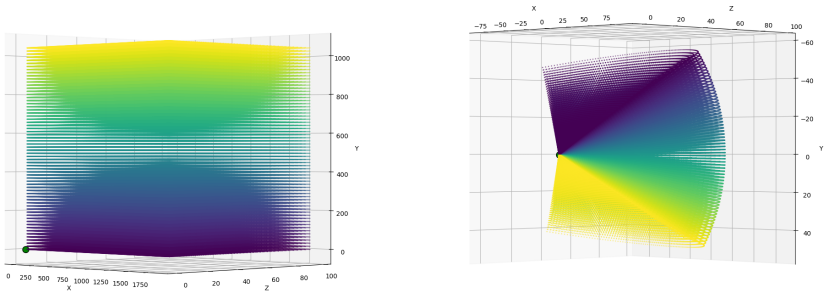


Figure 4.1: On the left is the original meshgrid, on the right is the transformed meshgrid after performing Kannala-unprojection.

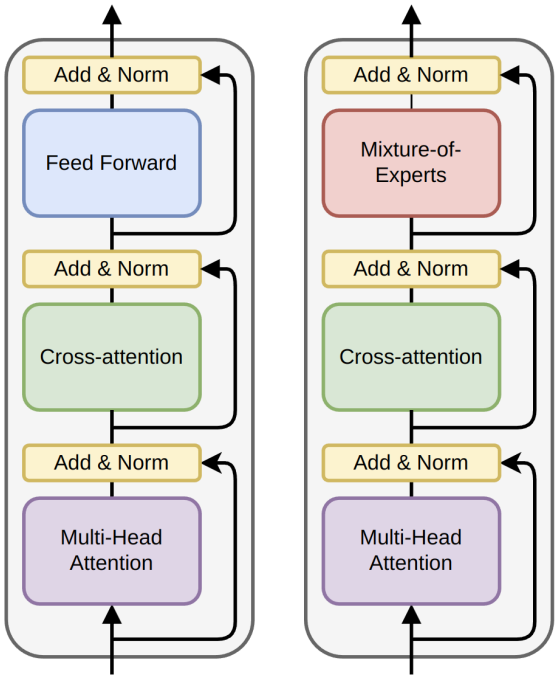


Figure 4.2: Comparison of two transformer decoder block architectures. The left block represents a standard transformer decoder with a feed-forward network (FFN) following the multi-head attention and cross-attention layers. The right block shows a modified architecture where the feed-forward network is replaced by a Mixture-of-Experts (MoE) layer, enhancing model capacity and sparsity. Both architectures use residual connections and layer normalization ("Add & Norm") after each sub-layer.

with replacing the FFN with an MoE layer. See Figure 4.2. The MoE layer provides a way to increase model capacity without sacrificing computational efficiency. This is achieved by activating neurons only sparsely.

Looking closer at the MoE layer, depicted in Figure 4.3, it consists of several key components: a gating function (router), multiple expert networks, and a mechanism for combining their outputs.

4.2.1 Gating Function

The gating function, or router, determines which experts are selected to process each input token. For a token embedding $x \in \mathbb{R}^d$, the router computes a vector of scores over N experts using a linear projection:

$$s = W_g x, \quad s \in \mathbb{R}^N \ , \tag{4.1}$$

where $W_g \in \mathbb{R}^{N \times d}$ is a learned weight matrix. These scores are converted into a probability distribution using the softmax function:

$$p = \text{softmax}(s). \tag{4.2}$$

Each p_i reflects the relative preference for assigning token x to expert i. In a sparse MoE, only a small number k of experts (typically $k \ll N$) are selected for each

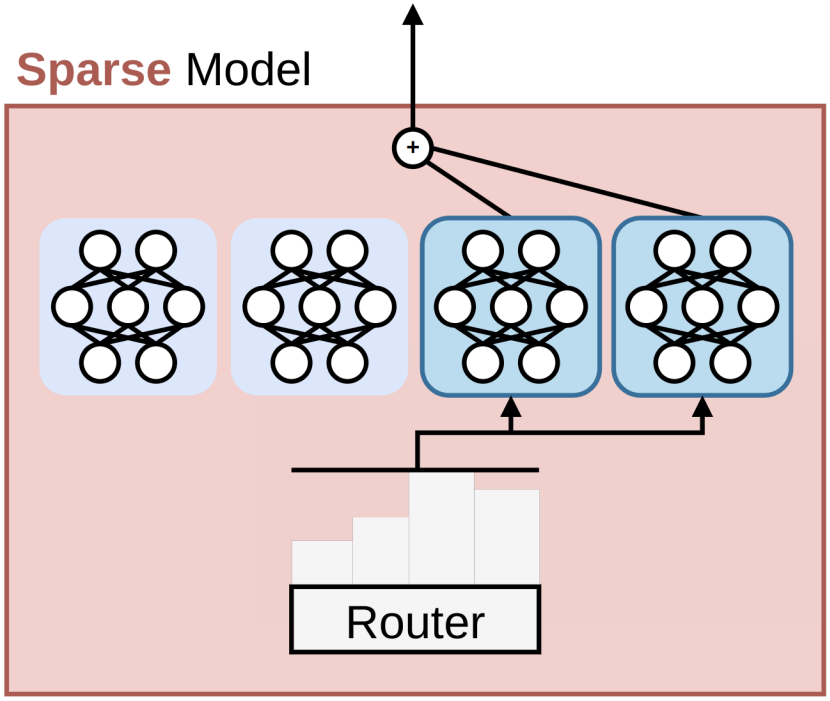


Figure 4.3: A sparse MoE layer. A router dynamically selects a subset of expert networks for each input token, activating only a few out of many available experts. The outputs of the selected experts are combined via a weighted sum to form the final output. This architecture allows the model to scale its capacity without increasing computational cost proportionally, enabling efficient training and inference.

token. We denote the indices of the selected experts as $\mathcal{I}(x)$:

$$\mathcal{I}(x) = \text{top}_k(p, k), \tag{4.3}$$

where $\text{top}_k(p, k)$ returns the indices of the k largest values in p. The corresponding gating weights $G(x) \in \mathbb{R}^N$ are then defined as:

$$G_i(x) = \begin{cases} p_i, & \text{if } i \in \mathcal{I}(x), \\ 0, & \text{otherwise.} \end{cases} \tag{4.4}$$

In some implementations, the non-zero entries in $G(x)$ are renormalized to sum to one. In our experiments, renormalization had little effect on performance, so we omit it for simplicity. The resulting gating weights are used to combine the outputs of the selected experts.

4.2.2 Experts

The experts are independent feed-forward networks (FFNs), each with distinct learned weights. When a token x is routed to an expert $i \in \mathcal{I}(x)$, it is processed as $E_i(x)$, where E_i denotes the ith expert. The final output of the MoE layer is the weighted sum of the selected expert outputs:

$$\text{Out}(x) = \sum_{i \in \mathcal{I}(x)} G_i(x) E_i(x). \tag{4.5}$$

This sparse activation allows the model to scale up its total number of parameters across all experts, while keeping the computational cost per token low. Since only k of N experts are active for each token, the overall efficiency remains comparable to a standard FFN.

4.2.3 Auxiliary Loss

An important aspect in training MoE models is ensuring that all experts are utilized effectively and that the load, i.e., number of tokens processed, is balanced across them. If the gating network consistently favors only a few experts, the benefits of having many experts diminishes. To encourage balanced load distribution, an auxiliary loss function is typically added to the main task loss during training.

We use the load-balancing loss as described in Fedus, Zoph, and Shazeer [10], repeated here for completeness. Given N experts indexed by $i = 1$ to N and a batch $\mathcal{B}$ with $\mathcal{T}$ tokens, the auxiliary loss is computed as the scaled dot-product between vectors f and P,

$$\text{Loss} = \alpha N \sum_{i=1}^{N} f_i P_i, \tag{4.6}$$

where α is hyperparameter, f_i is the fraction of tokens dispatched to expert i,

$$f_i = \frac{1}{T} \sum_{x \in \mathcal{B}} \mathbb{1} \left\{ \text{argmax}\, p(x) = i \right\} \tag{4.7}$$

and P_i is the fraction of router probability allocated for expert i,

$$P_i = \frac{1}{T} \sum_{x \in \mathcal{B}} p_i(x). \tag{4.8}$$

This auxiliary loss effectively prevents expert underutilization by ensuring that the router's probability distribution aligns with the actual token assignments.

5

Results

We evaluate several architectural variants of the PETR model on the ZOD. A dense transformer baseline is established, followed by sparse alternatives using MoE layers.

5.1 Dense Baseline

Our baseline non-MoE model uses a pretrained ResNet-50 backbone, 6 transformer decoder layers and $d_{\text{ffn}} = 2048$. All models are trained for 50 epochs with a similar setup as in Liu et al. [22], with an AdamW optimizer (weight decay=1e−2) and cosine annealing scheduler (initial learning rate 2e−4). The baseline model achieves an NDS score of 0.502 and a mAP score of 0.241. Figure 5.1 shows the loss graph and Figure 5.2 shows example predictions.

5.1.1 Activation Sparsity

Consistent with the lazy neuron phenomenon—where transformer FFNs exhibit increasing activation sparsity over time, as described by Li et al. [18]—we observe a similar trend in the original PETR model. As shown in Figure 5.3, we observe that between 90% and 95% of FFN activations yield zero across different decoder layers. This sparsity grows progressively deeper into the network, with later layers exhibiting particularly pronounced inactive pathways. The substantial computational resources allocated to these inactive components motivates the exploration of MoEs based architectures, where such sparsity can be exploited advantageously. Rather than maintaining uniformly active FFN pathways, MoEs naturally accommodate and benefit from activation sparsity by dynamically routing inputs to a subset of specialized expert subnetworks.

5.2 Evaluating the Mixture-of-Experts Variant

We now explore whether MoE layers can restore or even exceed the original model's performance. By swapping out dense FFN layers with sparse MoE layers, we aim to reintroduce model capacity in a more efficient way. Table 5.2 outlines the configurations evaluated. In all setups, we use the GeLU activation function and replace the FFN in every decoder block, i.e., placement frequency is 1/1. The main variables across these configurations are the number of active experts per input token (top-k

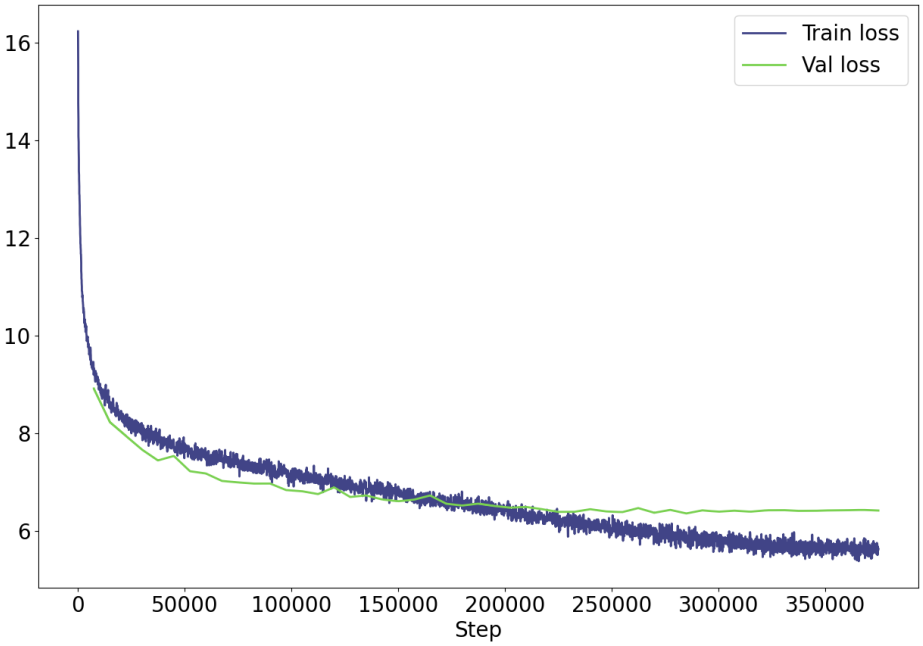


Figure 5.1: The loss graph of the baseline model.

Figure 5.2: Example image from the validation split of the ZOD dataset. Red boxes are the ground truth labels and green boxes are predictions by the baseline model.

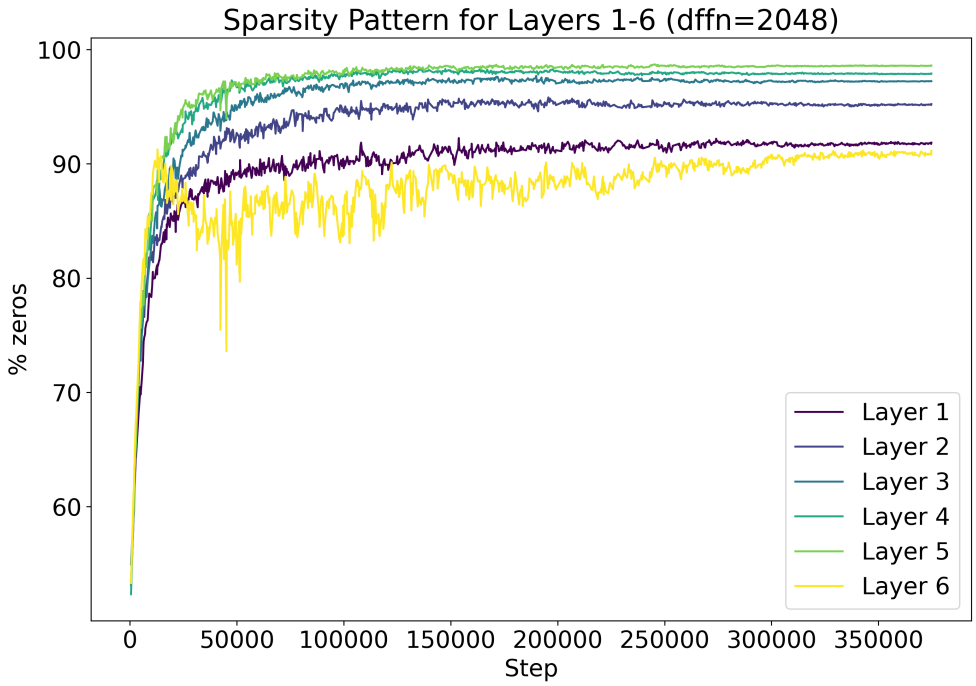


Figure 5.3: Activation sparsity across decoder layers during training of the baseline model, showing 90–95% zero activations in FFNs. Sparsity increases with network depth, with later layers exhibiting the most pronounced inactivity.

Table 5.1: Performance of PETR variants using MoE layers with different expert configurations. Each setup maintains the same number of active parameters as the dense baseline. Variants differ in the total expert pool size, and expert hidden dimensionality (d_{ffn}). All use load balancing weight $\alpha = 0.01$.

Experts (active/total)	**Neurons** (active/total)	d_{ffn}	NDS↑	mAP↑	mATE↓	mASE↓	mAOE↓
1/1	2048/2048	2048	0.502	0.242	0.0913	0.0263	0.0751
1/2	2048/4096	2048	0.500	0.239	0.0912	0.0264	0.0762
1/4	2048/8192	2048	0.500	0.239	0.0918	0.0264	0.0752
1/8	2048/16384	2048	0.500	0.239	0.0915	0.0264	0.0752
1/16	2048/32768	2048	0.500	0.239	0.0918	0.0265	0.0766
2/2	2048/2048	1024	0.503	0.243	0.0916	0.0263	0.0744
2/4	2048/4096	1024	0.502	0.241	0.0912	0.0263	0.0751
2/8	2048/8192	1024	0.502	0.241	0.0911	0.0263	0.0744
2/16	2048/16384	1024	0.501	0.240	0.0914	0.0264	0.0752

Table 5.2: Results with 2 active out of 8 total experts and varying load balancing loss weight α.

α	NDS↑	mAP↑	mATE↓	mASE↓	mAOE↓
0	0.502	0.241	0.0912	0.0263	0.0758
0.001	0.501	0.240	0.0915	0.0262	0.0751
0.01	0.502	0.241	0.0911	0.0263	0.0744
0.1	0.502	0.242	0.0913	0.0263	0.0750

selection), the total number of experts in the pool, and the hidden size of each expert network (d_{ffn}). Importantly, each configuration is designed to match the number of active parameters used in the original dense FFN baseline.

5.2.1 Expert Load Balancing

n important problem of a MoE layer is ensuring balanced token routing across experts to prevent expert collapse, where a small subset of experts dominates the computation. Without explicit regularization, the router often converges to a degenerate solution, underutilizing most experts. To demonstrate this, we analyze the distribution of routed tokens across experts both with and without the load-balancing loss. As shown in Figure 5.4, when trained without load-balancing loss, the token assignment becomes highly skewed, with a few experts processing the majority of tokens while others remain underused. In contrast, introducing the load-balancing loss encourages near-uniform routing, ensuring all experts get to contribute (Figure 5.5). This balanced utilization not only improves model capacity but also enhances robustness by distributing learning across the entire pool of experts. The results validate the necessity of the load-balancing term in maintaining efficient expert utilization and preventing degenerate routing behavior. Graphs for other values of α can be found in Appendix A.

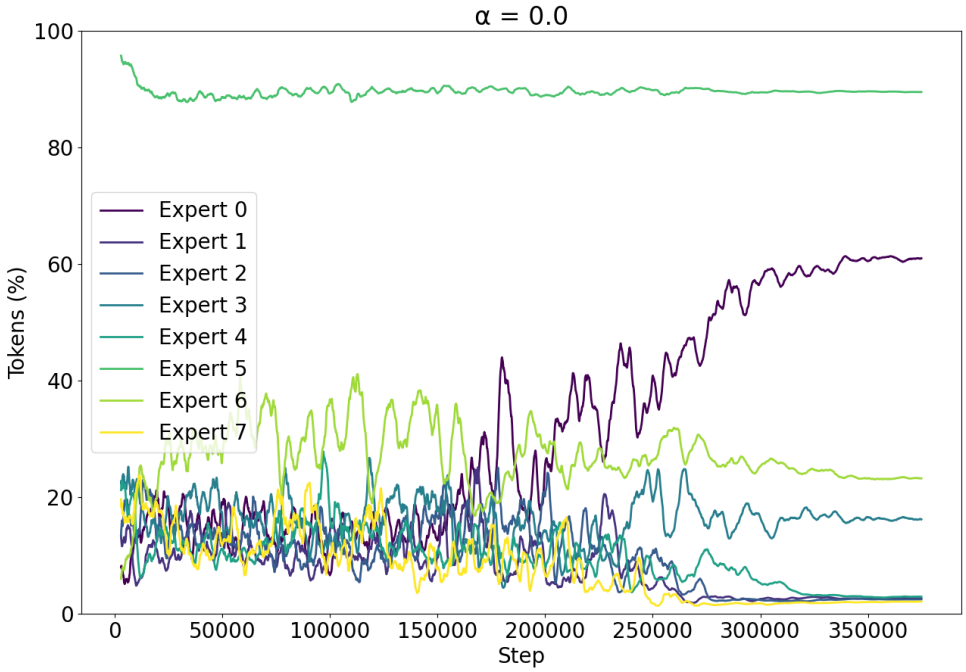


Figure 5.4: Expert utilization in layer 4 during training with 2 active out of 8 total experts and $\alpha = 0.0$. A few experts are used for the majority of tokens while some experts are almost never used. The rest of the layers look similar.

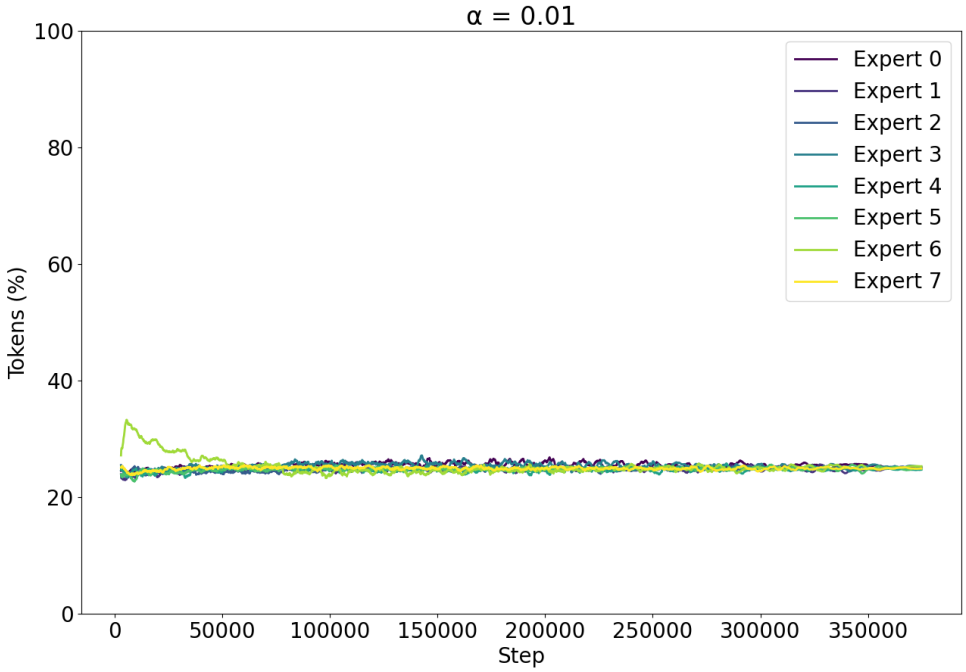


Figure 5.5: Expert utilization in layer 4 during training with 2 active out of 8 total experts and $\alpha = 0.01$. Each expert is used for around $2/8 = 25\%$ of tokens. The rest of the layers look similar.

Table 5.3: Performance of PETR with varying FFN dimensionality (d_{ffn}) on the ZOD dataset. The version with $d_{\text{ffn}} = 2048$ is the baseline.

d_{ffn}	**NDS↑**	**mAP↑**	**mATE↓**	**mASE↓**	**mAOE↓**
-	0.500	0.238	0.092	0.027	0.077
64	0.501	0.240	0.091	0.026	0.075
128	0.502	0.242	0.092	0.026	0.074
256	0.501	0.241	0.091	0.026	0.076
512	0.501	0.239	0.092	0.027	0.075
1024	0.502	0.241	0.091	0.026	0.075
2048	0.502	0.241	0.091	0.026	0.076
4096	0.500	0.238	0.092	0.026	0.075
8092	0.500	0.239	0.091	0.026	0.076

5.3 Varying d_{ffn}

Mixture of experts tries to simulate a larger d_{ffn} without requiring the compute of an actually larger d_{ffn}. However, since we did not see improved results, a natural question is whether there is a problem with MoE or if a larger d_{ffn} does not improve this model. Table 5.3 shows the evaluation metrics of the fully trained original model with varying d_{ffn}. Activation sparsity graphs similar to Figure 5.3 for other d_{ffn} configurations are provided in Appendix A.

Figure 5.6 presents a qualitative comparison of detection outputs on the ZOD dataset across three model variants: the baseline ($d_{\text{ffn}} = 2048$), a version without FFN, and a MoE-based variant. Surprisingly, all three variants achieve nearly identical performance, with no discernible qualitative or quantitative advantages observed across architectures. We have also tried reducing and removing the FFN in PETR for nuScenes and in Zenseact's internal proprietary model and dataset, and found similar results in both cases. In Zenseact's internal model, accuracy improved slightly when reducing d_{ffn} by a factor of 2 or 4, but worsened when removing it completely. However, the differences are small and it is unclear if they are statistically significant. This suggests that the choice of FFN configuration—whether removed entirely, kept at full capacity, or replaced with a MoE—has minimal impact on the model's ability to adapt to the ZOD benchmark. The consistent results imply that other architectural components or training dynamics may play a more significant role in shaping detection quality for this task.

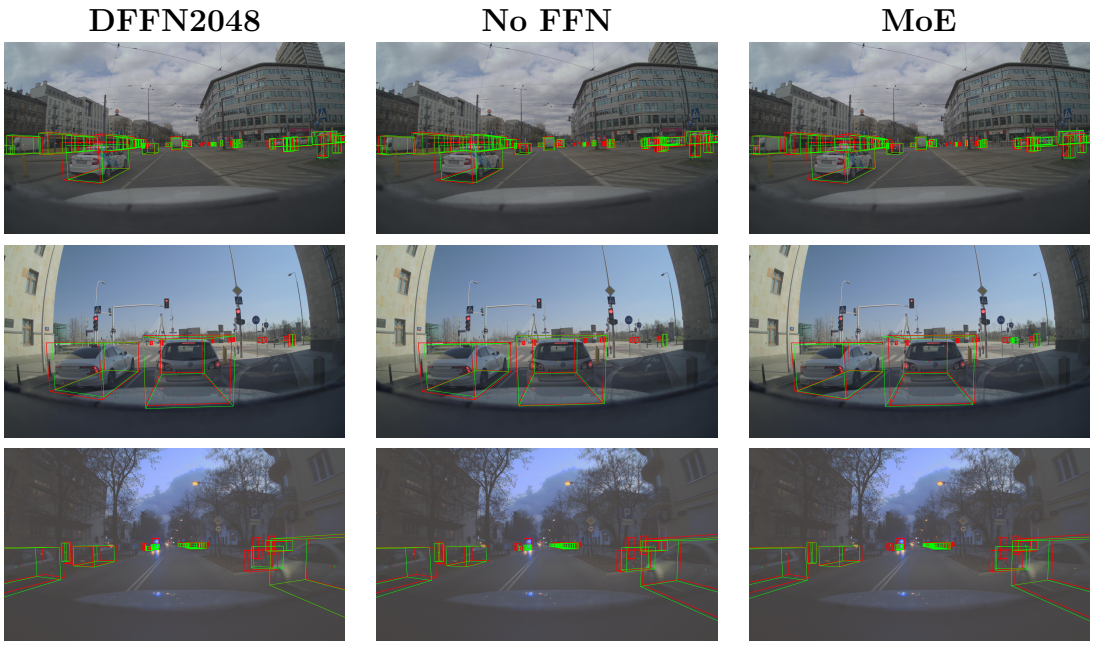


Figure 5.6: Qualitative comparison of detection outputs across model variants on ZOD dataset. Green boxes are predictions and red boxes are the ground truth labels.

6

Conclusion

This project set out to improve the efficiency and scalability of an existing transformer-based 3D object detection model by replacing the standard FFN component in the transformer-decoder with a MoE component. The idea was to take advantage of the conditional computation properties of MoEs to increase model capacity without incurring a proportional computational cost. Empirical results showed that the MoE substitution did not lead to measurable improvements in detection performance or efficiency. This outcome should not be interpreted as an inherent limitation of MoEs or a flaw in their integration. Further investigation revealed that the base model already possessed excessive model capacity. In fact, progressively reducing the hidden dimension of the FFN did not have an adverse effect on performance. Surprisingly, even when the FFN was completely removed, the model retained its performance level. We evaluated this phenomenon using our model with two established datasets, ZOD and nuScenes, as well as a proprietary model and dataset provided by Zenseact. In all cases, the results were highly consistent, suggesting that the phenomenon is not limited to a specific model or dataset. This finding suggests that the role of the FFN within the transformer-decoder is, in this context, functionally redundant. This finding challenges the common assumption that the FFN is an essential part of the transformer architecture. These insights not only clarify the outcome of the MoE experiment, but also raise broader questions about the necessity and role of FFNs in transformers. From a practical perspective, the implications are significant. Transformer blocks without FFNs are not just more parameter-efficient, they also reduce memory usage, speed up training, and improve inference times. Such improvements are especially relevant for real-time applications like within autonomous driving, where computational resources are scarce and require low latency.

6.1 Future Work

The unexpected redundancy of the FFN in the transformer decoder block, as observed in this work, raises several compelling directions for further research. While MoEs failed to improve performance despite significant added capacity, removing the FFN entirely also had negligible impact. This suggests that, at least for 3D object detection, the FFN may not be as essential as previously assumed. However, the extent of this redundancy remains an open question—does it arise due to the specific nature of the task, the model architecture, or a broader characteristic of perception

problems? To explore this, it is necessary to investigate related tasks—such as object tracking, instance segmentation, and occupancy prediction—across a range of model architectures and datasets. If FFNs consistently appear redundant in these domains, this could indicate a more general pattern, opening the door to simpler transformer designs that rely solely on attention mechanisms. On the other hand, if performance degrades in some tasks when the FFN is removed, this would suggest that FFNs play a more specialized role under certain conditions—such as refining features when input signals are more ambiguous or enabling more complex forms of reasoning. In contrast, their redundancy in other tasks may point to contexts where attention alone, possibly in combination with strong backbone features, is sufficient. Either outcome would contribute towards our understanding of the role of the FFN.

There is also a broader question of whether the FFN's importance is domain specific. For example, in NLP, FFNs are considered essential. If FFNs prove largely redundant in vision-based models but crucial in language models, this could suggest fundamental differences in the effect of attention across modalities. A hypothesis is that perception tasks benefit more from geometric reasoning—relying on features that may already be well-captured by CNN backbones in combination with attention layers—whereas NLP tasks require nonlinear transformations, which the FFN provides.

Ultimately, the findings of this work hint at the possibility of rethinking the design of transformers within perception systems. Preliminary experiments show that the transformer size can also be reduced in some other ways (for example the number of layers) without much impact. One possible explanation is that most of the processing takes place in the CNN. This raises the question of how much the capacity of the sensor fusion and prediction generation network can be reduced. Could some kind of linear model suffice?

6.2 Ethics

The ethical and sustainability aspects are especially important in the context of using AI within autonomous systems such as in autonomous vehicles. A central ethical question is the impact on decision-making. Autonomous vehicles are typically dependent on object identification systems for making real-time decisions that directly affect human safety. If those systems are heavily reliant on AI, it is important to ensure that they are secure, reliable, and thoroughly tested. While explainability—the ability to understand and interpret AI decision-making processes—can be valuable for trust and accountability, it is not always a strict requirement for safety. Some "black-box" solutions may prove to be safe and trustworthy through extensive real-world usage and rigorous validation over time. Regulatory frameworks play a crucial role in ensuring the safety and reliability of AD/ADAS. Standards such as ISO 26262, which focuses on functional safety of road vehicles, and assessment programs like Euro NCAP, which evaluates vehicle safety performance, provide systematic approaches for establishing a baseline for safety.

Another important aspect that has become more and more important since the

AI boom is the energy efficiency of AI systems within the context of sustainability. Training and deploying large models often requires significant computational resources, contributing to high energy consumption. Optimizing models to make them more resource efficient can help make them more environmentally friendly. Furthermore, optimized resource efficient models would also lower the requirements on hardware in the sense that they could be deployed on low-power devices, reducing hardware waste and promoting accessibility.

Bibliography

[1] Joshua Ainslie et al. "GQA: Training Generalized Multi-Query Transformer Models from Multi-Head Checkpoints". In: *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*. Ed. by Houda Bouamor, Juan Pino, and Kalika Bali. Singapore: Association for Computational Linguistics, Dec. 2023, pp. 4895–4901. DOI: 10.18653/v1/2023.emnlp-main.298. URL: https://aclanthology.org/2023.emnlp-main.298/.

[2] Mina Alibeigi et al. "Zenseact Open Dataset: A Large-Scale and Diverse Multimodal Dataset for Autonomous Driving". In: *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*. Oct. 2023, pp. 20178–20188.

[3] Holger Caesar et al. "nuScenes: A Multimodal Dataset for Autonomous Driving". In: *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2020.

[4] Weilin Cai et al. *A Survey on Mixture of Experts*. 2024. arXiv: 2407.06204 [cs.LG]. URL: https://arxiv.org/abs/2407.06204.

[5] Nicolas Carion et al. "End-to-End Object Detection with Transformers". In: *Computer Vision – ECCV 2020*. Ed. by Andrea Vedaldi et al. Cham: Springer International Publishing, 2020, pp. 213–229. ISBN: 978-3-030-58452-8.

[6] Bowen Cheng et al. "Masked-Attention Mask Transformer for Universal Image Segmentation". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2022, pp. 1290–1299.

[7] Róbert Csordás, Kazuki Irie, and Jürgen Schmidhuber. "Approximating Two-Layer Feedforward Networks for Efficient Transformers". In: *Findings of the Association for Computational Linguistics: EMNLP 2023*. Ed. by Houda Bouamor, Juan Pino, and Kalika Bali. Singapore: Association for Computational Linguistics, Dec. 2023, pp. 674–692. DOI: 10.18653/v1/2023.findings-emnlp.49. URL: https://aclanthology.org/2023.findings-emnlp.49/.

[8] DeepSeek-AI et al. *DeepSeek-V2: A Strong, Economical, and Efficient Mixture-of-Experts Language Model*. 2024. arXiv: 2405.04434 [cs.CL]. URL: https://arxiv.org/abs/2405.04434.

[9] DeepSeek-AI et al. *DeepSeek-V3 Technical Report*. 2024. arXiv: 2412.19437 [cs.CL]. URL: https://arxiv.org/abs/2412.19437.

[10] William Fedus, Barret Zoph, and Noam Shazeer. "Switch Transformers: Scaling to Trillion Parameter Models with Simple and Efficient Sparsity". In:

Journal of Machine Learning Research 23.120 (2022), pp. 1–39. URL: http://jmlr.org/papers/v23/21-0998.html.

[11] Kaiming He et al. "Deep Residual Learning for Image Recognition". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2016.

[12] Robert A. Jacobs et al. "Adaptive Mixtures of Local Experts". In: *Neural Computation* 3.1 (Mar. 1991), pp. 79–87. ISSN: 0899-7667. DOI: 10.1162/neco.1991.3.1.79. eprint: https://direct.mit.edu/neco/article-pdf/3/1/79/812104/neco.1991.3.1.79.pdf. URL: https://doi.org/10.1162/neco.1991.3.1.79.

[13] J. Kannala and S.S. Brandt. "A generic camera model and calibration method for conventional, wide-angle, and fish-eye lenses". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 28.8 (2006), pp. 1335–1340. DOI: 10.1109/TPAMI.2006.153.

[14] Jared Kaplan et al. "Scaling laws for neural language models". In: *arXiv preprint arXiv:2001.08361* (2020). URL: https://arxiv.org/abs/2001.08361.

[15] Angelos Katharopoulos et al. "Transformers are RNNs: Fast Autoregressive Transformers with Linear Attention". In: *Proceedings of the 37th International Conference on Machine Learning*. Ed. by Hal Daumé III and Aarti Singh. Vol. 119. Proceedings of Machine Learning Research. PMLR, 13–18 Jul 2020, pp. 5156–5165. URL: https://proceedings.mlr.press/v119/katharopoulos20a.html.

[16] Woosuk Kwon et al. "Efficient Memory Management for Large Language Model Serving with PagedAttention". In: *Proceedings of the 29th Symposium on Operating Systems Principles*. SOSP '23. Koblenz, Germany: Association for Computing Machinery, 2023, pp. 611–626. ISBN: 9798400702297. DOI: 10.1145/3600006.3613165. URL: https://doi.org/10.1145/3600006.3613165.

[17] Mike Lewis et al. "BASE Layers: Simplifying Training of Large, Sparse Models". In: *Proceedings of the 38th International Conference on Machine Learning*. Ed. by Marina Meila and Tong Zhang. Vol. 139. Proceedings of Machine Learning Research. PMLR, 18–24 Jul 2021, pp. 6265–6274. URL: https://proceedings.mlr.press/v139/lewis21a.html.

[18] Zonglin Li et al. "The Lazy Neuron Phenomenon: On Emergence of Activation Sparsity in Transformers". In: *The Eleventh International Conference on Learning Representations*. 2023. URL: https://openreview.net/forum?id=TJ2nxciYCk-.

[19] Tianyang Lin et al. "A survey of transformers". In: *AI Open* 3 (2022), pp. 111–132. ISSN: 2666-6510. DOI: https://doi.org/10.1016/j.aiopen.2022.10.001. URL: https://www.sciencedirect.com/science/article/pii/S2666651022000146.

[20] Tsung-Yi Lin et al. "Focal Loss for Dense Object Detection". In: *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*. Oct. 2017.

[21] Ruijin Liu et al. “End-to-End Lane Shape Prediction With Transformers”. In: *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)*. Jan. 2021, pp. 3694–3702.

[22] Yingfei Liu et al. “PETR: Position Embedding Transformation for Multi-view 3D Object Detection”. In: *Computer Vision – ECCV 2022*. Ed. by Shai Avidan et al. Cham: Springer Nature Switzerland, 2022, pp. 531–548. ISBN: 978-3-031-19812-0.

[23] Niklas Muennighoff et al. *OLMoE: Open Mixture-of-Experts Language Models*. 2024. arXiv: 2409.02060 [cs.CL]. URL: https://arxiv.org/abs/2409.02060.

[24] Joan Puigcerver et al. “From Sparse to Soft Mixtures of Experts”. In: *The Twelfth International Conference on Learning Representations*. 2024. URL: https://openreview.net/forum?id=jxpsAj7ltE.

[25] Carlos Riquelme et al. “Scaling Vision with Sparse Mixture of Experts”. In: *Advances in Neural Information Processing Systems*. Ed. by M. Ranzato et al. Vol. 34. Curran Associates, Inc., 2021, pp. 8583–8595. URL: https://proceedings.neurips.cc/paper_files/paper/2021/file/48237d9f2dea8c74c2a72126cf6
Paper.pdf.

[26] Noam Shazeer et al. “Outrageously Large Neural Networks: The Sparsely-Gated Mixture-of-Experts Layer”. In: *International Conference on Learning Representations*. 2017. URL: https://openreview.net/forum?id=B1ckMDqlg.

[27] Yi Tay et al. “Efficient Transformers: A Survey”. In: *ACM Comput. Surv.* 55.6 (Dec. 2022). ISSN: 0360-0300. DOI: 10.1145/3530811. URL: https://doi.org/10.1145/3530811.

[28] Ashish Vaswani et al. “Attention is All you Need”. In: *Advances in Neural Information Processing Systems*. Ed. by I. Guyon et al. Vol. 30. Curran Associates, Inc., 2017. URL: https://proceedings.neurips.cc/paper_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf.

[29] Tianwen Wei et al. *Skywork-MoE: A Deep Dive into Training Techniques for Mixture-of-Experts Language Models*. 2024. arXiv: 2406.06563 [cs.CL]. URL: https://arxiv.org/abs/2406.06563.

[30] Manzil Zaheer et al. *Big Bird: Transformers for Longer Sequences*. 2021. arXiv: 2007.14062 [cs.LG]. URL: https://arxiv.org/abs/2007.14062.

[31] Sixiao Zheng et al. “Rethinking semantic segmentation from a sequence-to-sequence perspective with transformers”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)* (2021), pp. 6881–6890. URL: https://openaccess.thecvf.com/content/CVPR2021/html/Zheng_Rethinking_Semantic_Segmentation_From_a_Sequence-to-Sequence_Perspective_With_Transformers_CVPR_2021_paper.html.

[32] Xizhou Zhu et al. “Deformable DETR: Deformable transformers for end-to-end object detection”. In: *Proceedings of the International Conference on Learning Representations (ICLR)* (2021). URL: https://openreview.net/forum?id=gZ9hCDWe6ke.

A

Appendix 1

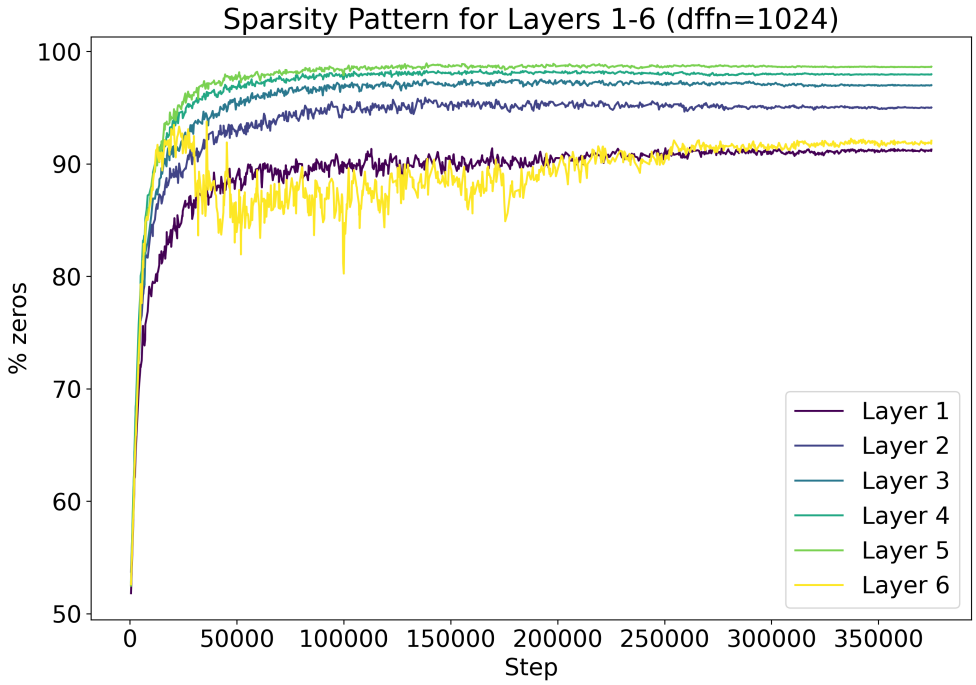


Figure A.1: Activation sparsity across decoder layers during training of the original PETR model ($d_{\text{ffn}} = 1024$, showing 90–95% zero activations in FFNs. Sparsity increases with network depth, with later layers exhibiting the most pronounced inactivity.

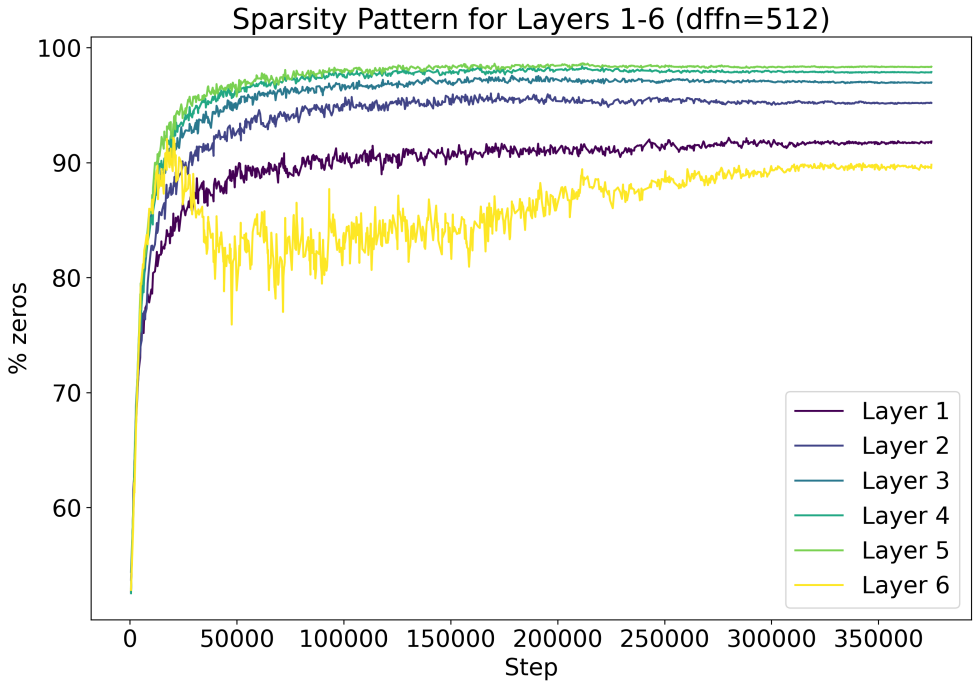


Figure A.2: Activation sparsity across decoder layers during training of the original PETR model ($d_{\text{ffn}} = 512$, showing 90–95% zero activations in FFNs. Sparsity increases with network depth, with later layers exhibiting the most pronounced inactivity.

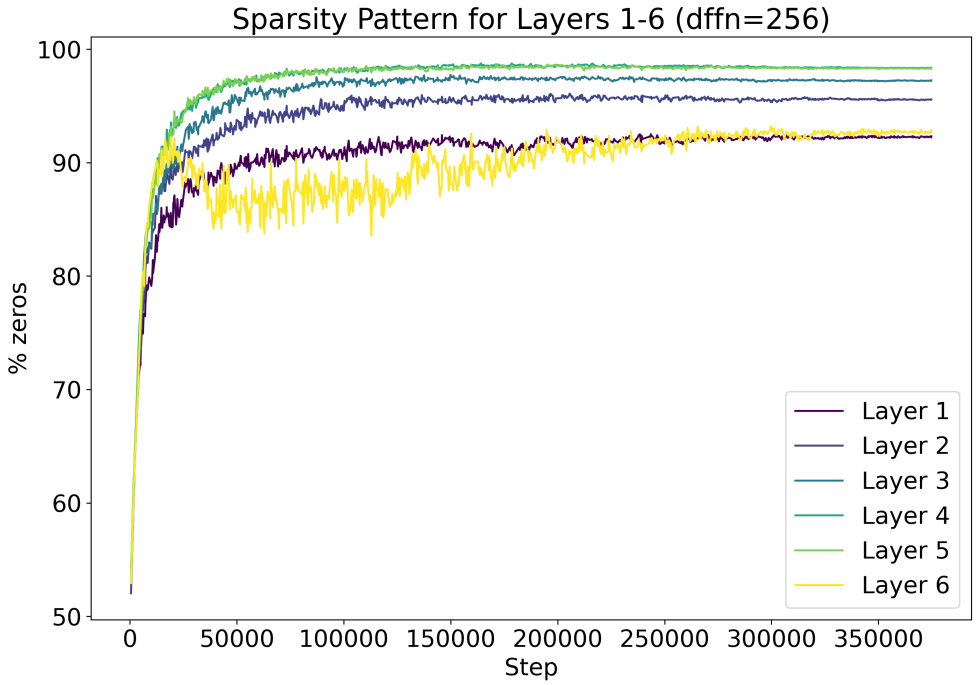


Figure A.3: Activation sparsity across decoder layers during training of the original PETR model ($d_{\text{ffn}} = 256$, showing 90–95% zero activations in FFNs. Sparsity increases with network depth, with later layers exhibiting the most pronounced inactivity.

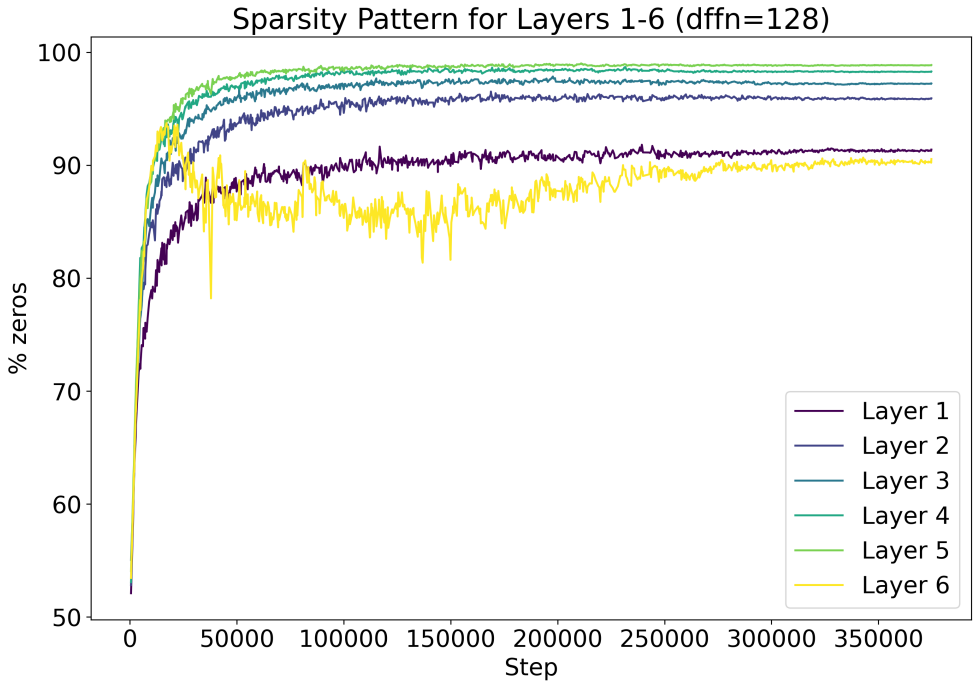


Figure A.4: Activation sparsity across decoder layers during training of the original PETR model ($d_{\text{ffn}} = 128$, showing 90–95% zero activations in FFNs. Sparsity increases with network depth, with later layers exhibiting the most pronounced inactivity.

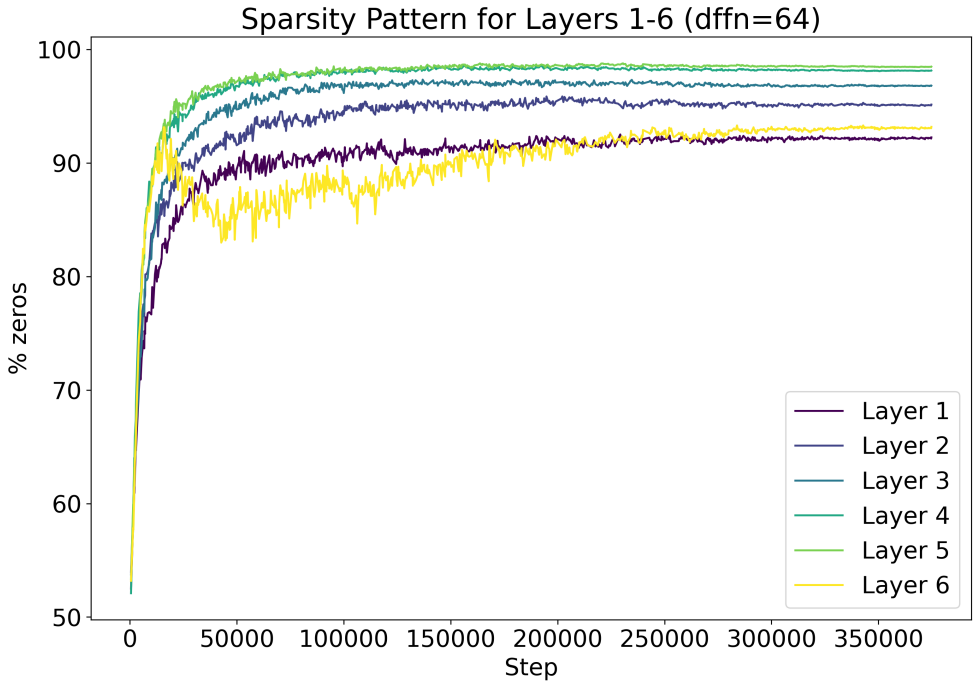


Figure A.5: Activation sparsity across decoder layers during training of the original PETR model ($d_{\text{ffn}} = 64$, showing 90–95% zero activations in FFNs. Sparsity increases with network depth, with later layers exhibiting the most pronounced inactivity.

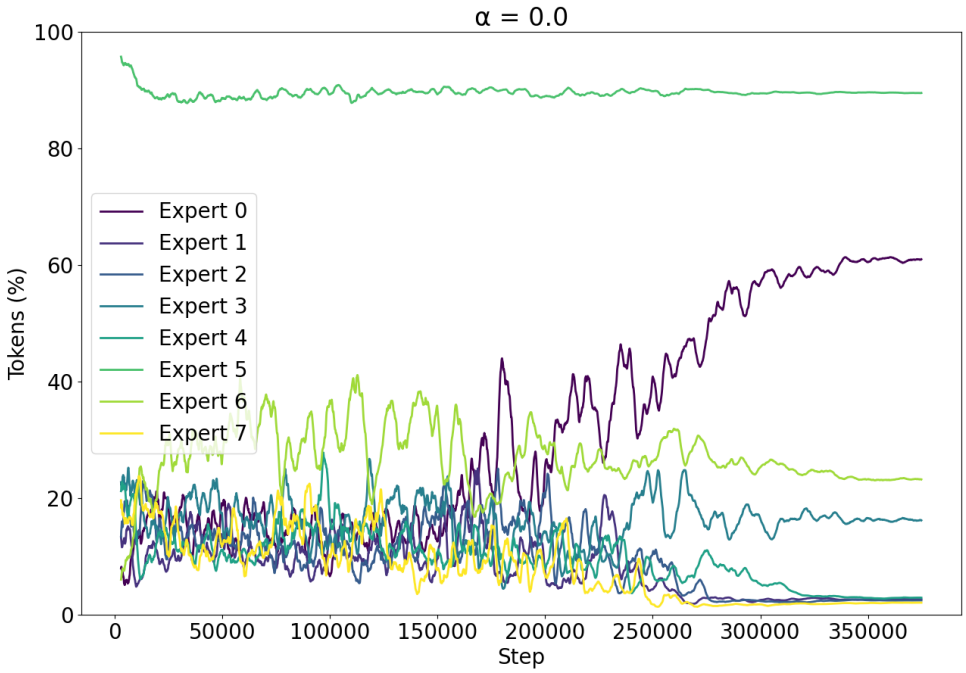


Figure A.6: Expert utilization in layer 4 during training with 2 active out of 8 total experts and $\alpha = 0$.

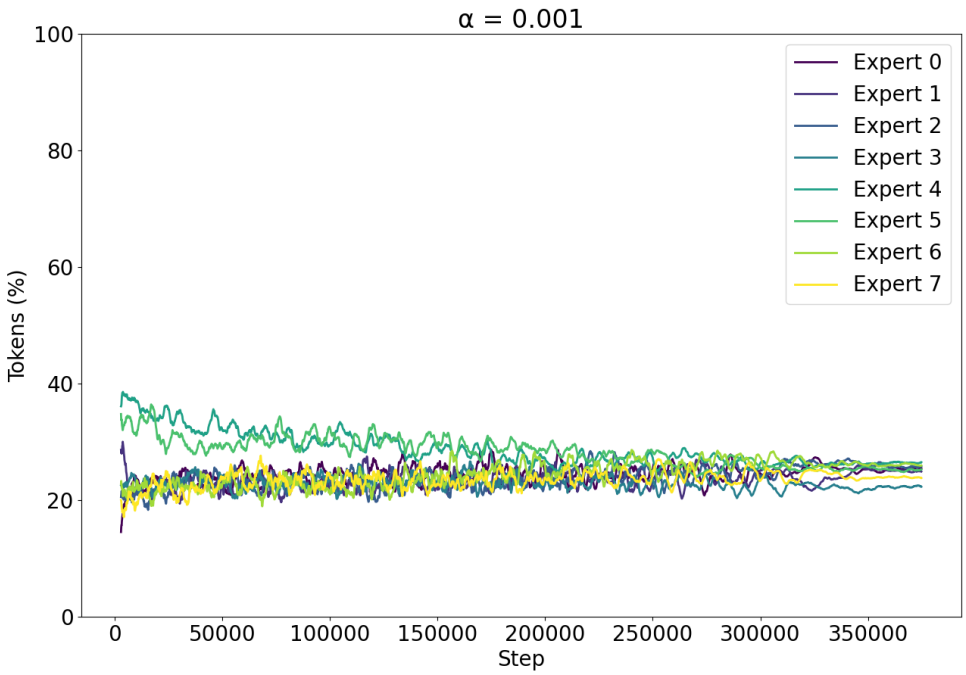


Figure A.7: Expert utilization in layer 4 during training with 2 active out of 8 total experts and $\alpha = 0.001$.

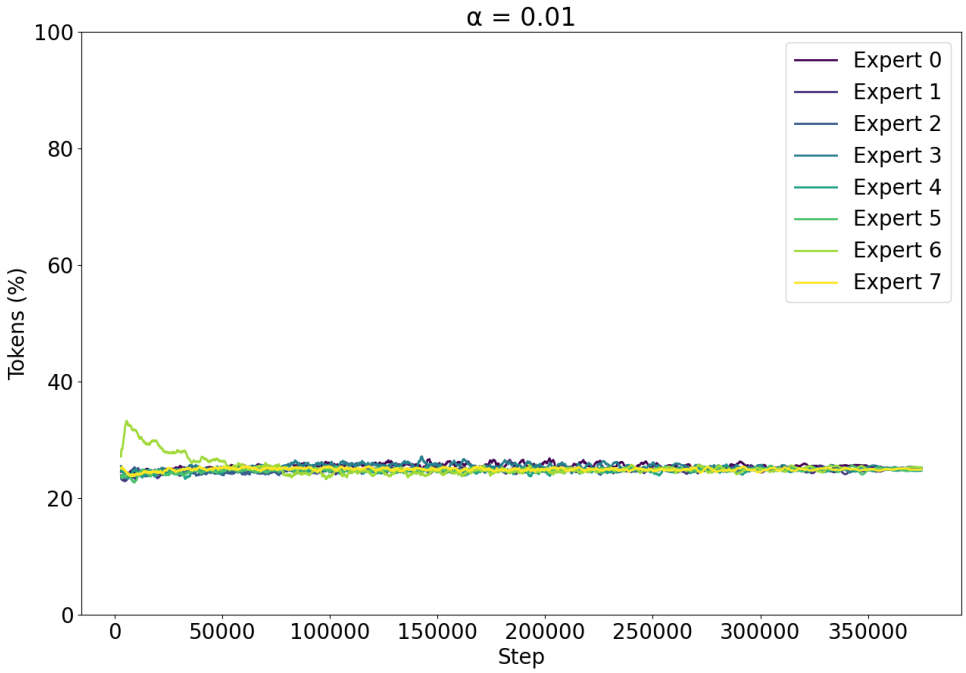


Figure A.8: Expert utilization in layer 4 during training with 2 active out of 8 total experts and $\alpha = 0.01$.

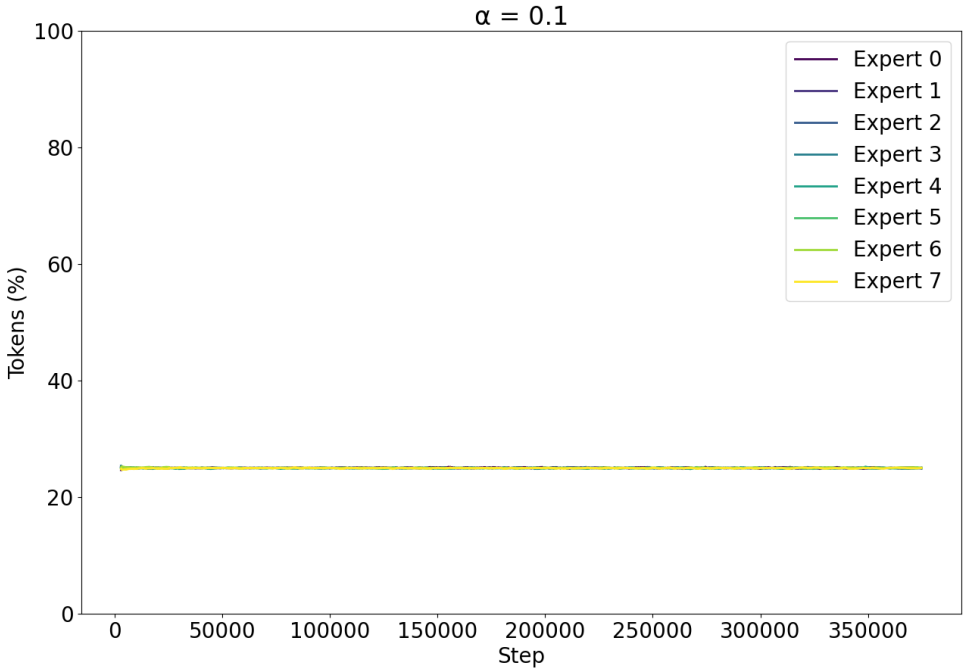


Figure A.9: Expert utilization in layer 4 during training with 2 active out of 8 total experts and $\alpha = 0.1$.

DEPARTMENT OF ELECTRICAL ENGINEERING
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden
www.chalmers.se

CHALMERS
UNIVERSITY OF TECHNOLOGY