SWAN: World-Aware Adaptive Multimodal Networks for Runtime Variations

Jason Wu[1], Shir-Kang Scott Jin[2], Yuyang Yuan[1], Maggie Wigness[3], Lance M. Kaplan[3], Hang Qiu[2], and Mani Srivastava[1]

[1] University of California, Los Angeles
[2] University of California, Riverside
[3] U.S. Army DEVCOM Army Research Laboratory

Abstract. Multimodal deep neural networks deployed in realistic environments must contend with runtime variations: changes in modality quality, overall input complexity, and available platform resources. Current networks struggle with such fluctuations – adaptive networks cannot adhere to a strict compute budget, controller-based networks neglect to consider input complexity, and statically provisioned networks fail at all the above. Consequently, they do not extract maximum utility from the expended computational resources. We present SWAN (**S**ample and **W**orld-**A**ware Multimodal **N**etwork), the first adaptive multimodal network that accomplishes all three goals. SWAN employs a quality-aware controller to assign resources among modalities according to a variable user-specified maximum budget. Within this budget, an adaptive gating module further optimizes efficiency by scaling layer utilization according to sample complexity. For further gains, SWAN also employs a token dropping module that masks semantically irrelevant multimodal features before performing detections. We evaluate SWAN in the domain of autonomous driving with complex multi-object 3D detection, reducing FLOPs by up to 49% with minimal degradation.

Keywords: Multimodal Learning · Neural Network Adaptation · Computational Efficiency

1 Introduction

Real world systems frequently contend with *runtime variations* as the world changes during system operation. A camera-based object detector may encounter varying environmental conditions that drastically alter the utility of its input modality. Similarly, a system deployed on a mobile robot will encounter diverse computational resource availability owing to thermal throttling or power limitations. Despite the large class of runtime variations, the most prevalent variations can be largely grouped into three categories – input modality quality of information (QoI) (*e.g.* sensor failure, dynamic weather), sample complexity (*e.g.* more or less sensing targets), and platform dynamics (*e.g.* thermal throttling).

A system's ability to adjust to such variations determines its utility in realistic deployments. One particularly relevant scenario involves the broad space of

autonomous vehicles (AVs), ranging from small mobile robots to road-certified vehicles. The majority of AVs will encounter different environmental lighting or weather, and must exhibit robustness to resulting variations in input QoI. The AV's platform capabilities can also vary, where faster traveling speeds, low battery capacity, or thermal throttling can affect the amount of computation available for the current input sample. Finally, adapting computation based on sample complexity is also an important consideration. Since AVs operate on edge devices with limited resources, reducing computation on "easier" samples can translate to meaningful reductions in latency and energy consumption. The inability to adapt to any one of these variations can severely hamper the safety or utility of a given AV.

Existing works often deal with each of these variations separately. First, deep multimodal neural networks have emerged as a promising method for combating variable QoI. By virtue of aggregating information across multimodal sensors, such networks mitigate environmental corruptions in the input data through redundancy. Thus, multimodal networks are the de facto standard in AVs [1,20,28]. Although they provide robustness against variable QoI, the additional complexity of these networks can complicate adaptation to other runtime variations. Next, to meet dynamic computational budgets, reconfigurable neural networks modulate network computation during inference time. Once-For-All [6], LayerDrop [8], and ADMN [27] train adaptive networks where modules or layers can selectively be omitted at inference time for budget flexibility. Finally, input-adaptive systems incorporate knowledge of the complexity of the input to flexibly increase or reduce the computation performed on each sample [12,15,23]. For instance, AdaViT [23] dynamically removes attention heads, tokens, and layers from the network according to the nature of the input.

However, current literature falls short of proposing a system that *jointly addresses* these dynamics. The key difficulty lies in the compounding complexity that arises when these strategies are combined. To illustrate, once multimodality is introduced to combat dynamic QoI, adapting network utilization to input complexity requires consideration of shared information across modalities. Similarly, techniques that adapt network computation to a maximum budget are difficult to incorporate with those implementing per-sample complexity adaptation. We propose `SWAN`, an adaptive multimodal network that addresses all three runtime variations – changes in modality QoI, maximum compute, and sample complexity. Although this problem is universal across any domain deploying multimodal neural networks in real-world environments, we focus specifically on autonomous vehicles (AVs) due to the task complexity (multimodal, multi-object 3D detection), robust evaluation frameworks, and high quality datasets.

First, we train a *layer-wise adaptive multimodal network* based on the existing CMT [28] AV detection framework to allow for flexible allocation of layers during inference-time. Next, we introduce a controller that determines the optimal allocation of layers across modalities for the current budget and relative modality QoI. The controller is trained through *NeuralSort* [10] gradient approximation, enabling an understanding of relative importance among layers

which existing controller-based techniques [27] fail to model. Finally, SWAN also minimizes resource usage within the user-specified maximal budget, enabling additional reductions in compute and latency. We present two complementary innovations – a feature-aware *SkipGate* module that conditionally executes the next layer, and a token dropout mechanism that filters out redundant tokens before the detection head. Both methods are conditioned upon the input, enabling efficiency on simpler inputs while retaining flexibility on more complex samples.

We evaluated SWAN on the popular nuScenes dataset [5] with modality QoI degradation simulated through the MultiCorrupt [2] pipeline. Across various maximum allowable budgets, SWAN outperforms related baselines by up to 9 NDS and 14.5 mAP while achieving up to a 49% reduction in FLOPs compared to a fully-provisioned network. SWAN's controller correctly prioritizes high-QoI modalities under compute constraints, while the SkipGate and token pruning modules significantly reduce network computation. Additionally, we benchmarked SWAN on an Nvidia Jetson Orin edge device and with real-world data to further demonstrate its practicality. We provide the code at https://github.com/nesl/SWAN.

In summary, our main novel contributions are as follows:

1. We introduce the first multimodal network that modulates resource allocation across modalities in accordance to modality QoI, sample complexity, and a user-defined maximum budget
2. SWAN's lightweight, QoI-aware controller selects an optimal layer configuration given a specified budget, while retaining end-to-end differentiability through the *NeuralSort* gradient estimation technique
3. We additionally integrate a feature-based layer dropping module (*SkipGate*) that is conditioned on the output of the controller, and filter out tokens prior to the detection head for further efficiency gains
4. We perform evaluations on synthetically corrupted and real-world data, while also running on edge hardware to verify efficiency gains

2 Related Work

Multimodal AV Neural Networks. Owing to strict robustness and accuracy demands of autonomous driving, many AV perception stacks rely on multimodal deep neural networks [1,7,13,14,20]. For example, BEVFusion [20] combines information across LiDAR and camera in the BEV space, improving detection performance by over 19 NDS and 5 NDS relative to the best camera and LiDAR unimodal detectors, respectively. Additionally, these multimodal networks also showcase greater robustness to environmental noise when compared to their unimodal counterparts [1]. Adapting these multimodal networks to runtime variations that AVs commonly encounter is an open challenge.

Input-Aware Deep Neural Networks. Input-aware deep networks modify the network behavior according to the complexity of the input – saving layers on simpler inputs while leveraging full network capacity for more difficult samples [4,12,16,23,26]. AdaVIT [23] introduces a highly flexible ViT in which attention

heads, layers and patches are flexibly executed according to input complexity. Early-Exit techniques [4,12,16] terminate execution once a confidence threshold is reached, enabling reduced computation on simpler inputs. These networks are incompatible with the strict limits on allowable computation that can occur in AV settings. Existing adaptive networks optimize the *average case computation* with no guarantees on worse-case.

Adaptive Multimodal Networks. Although the majority of adaptive networks are unimodal, there exist a few adaptive *multimodal* networks. Several recent works [11,17,25] explored adaptive selection strategies, choosing to activating modalities or neural experts based on the input. ADMN [27] presents a more granular adaptation with a QoI-aware controller activating layers within a single multimodal network to meet a specified budget. However, they neglect to consider adaptation *within* a particular compute budget, while also being limited to toy networks with simple objectives (*e.g.* single object localization). Conversely, `SWAN` employs a flexible, budget-aware adaptation policy in the complex realm of AVs.

3 Methodology

`SWAN` accomplishes three objectives: it adapts the network to a user-specified maximum budget, decides an optimal allocation of resources among modalities for that budget, and then minimizes resource usage according to the input. In AV scenarios, this is particularly critical as available computational resources change over time (*e.g.* thermal throttling, energy constraints), coupled with rapidly changing modality QoI (*e.g.* changing environment).

Fig. 1 showcases the overall architecture of `SWAN`. Aside from the standard neural components present in most AV detection networks (discussed in the next section), we introduce three new components. First, `SWAN` learns a *QoI-aware controller* that intelligently assigns layers to modality backbones according to their marginal utility. The controller is trained to accommodate a diverse set of layer budgets, and will maximize its usage of a specified budget at inference time. Next, to ensure that valuable resources are not needlessly consumed if the budget is set too high, we integrate our *SkipGate* module into the multimodal network. Based on the feature vectors output by the prior layer and additional contextual information, SkipGate will determine whether to execute the current layer, enabling a reduction of computation within the outer controller's allocation. Finally, since not all areas of the input are equally important to the detection task, we implement a *layer-dropping* algorithm prior to the detection head that focuses computation on the semantically meaningful tokens.

3.1 Layer-Adaptive Multimodal AV Network

Architecture. `SWAN` is built upon the CMT [28] multimodal camera and LiDAR architecture. CMT processes each modality with a modality specific network, and adds 3D positional encodings to the unimodal features prior to fusion with

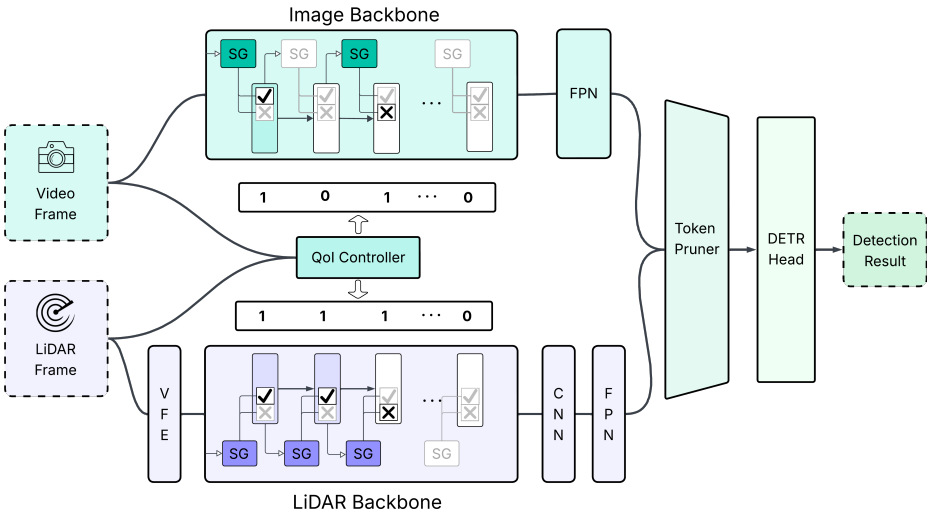

Fig. 1: SWAN architecture. The QoI controller allocates resources among the backbones according to the input QoI and the current budget. For each selected layer, a modality-shared SkipGate module decides whether to actually execute the layer. The unimodal features are filtered by a token pruning module before the detection head.

a transformer decoder. We select the Swin-Tiny ViT [19] as the camera encoder and FlatFormer Transformer [21] as the LiDAR encoder. Following FlatFormer and CMT, we utilize a Dynamic Voxel Feature Encoder (VFE), a SECOND-based convolutional backbone to process our dense LiDAR features, and independent feature pyramid networks to merge multi-scale features (Fig. 1). The bulk of the computation occurs in the transformer encoders; these auxiliary networks are lightweight and are not targeted for adaptation.

Training Procedure. CMT presents a multimodal fusion framework for AV detection and does not concern itself with flexible runtime adaptation. We incorporate LayerDrop [8] into network training by introducing a layer dropout rate (*i.e.* 0.2) within FlatFormer and Swin backbones. Over the training process, each layer is stochastically dropped out according to the rate, thereby conditioning the multimodal network to function with a subset of backbone layers. We provide further details on the training process in the Appendix.

3.2 QoI-Aware Controller

Controller Architecture. The controller allocates a specified budget of layers across the adaptive-depth backbones in the multimodal network. To ensure proper allocation, the controller must be aware of the relative input modality QoI. In Fig. 2, we provide a detailed overview of the controller.

First, the controller learns to identify the QoI characteristics of the input samples according to Eq. (1):

$$\mathbf{z} = \text{Concat}_{m \in M}(f_{\theta_m}(\mathbf{x}_m)) \; ; \; \mathcal{L}_{\text{env}} = \text{CE}(\text{MLP}_{\text{env}}(\mathbf{z}), y_{\text{env}}) \tag{1}$$

where f_{θ_m} is a lightweight convolutional network for modality $\mathbf{x}_m$, y_{env} is the ground truth corruption label (*e.g.* darkness), and CE is the cross-entropy loss. $\mathcal{L}_{env}$ ensures the information within $\mathbf{z}$ represents the QoI information.

Next, the controller maintains a library of fixed sinusoidal positional embeddings $\mathcal{E} = \{\mathbf{e}_b \in \mathbb{R}^d \mid b \in \mathcal{B}\}$ corresponding to a set of N user-specified layer budgets $\mathcal{B} = \{b_1, b_2, \ldots, b_N\}$. During training, we sample a budget $b \sim \text{Uniform}(\mathcal{B})$ for each input, ensuring that the controller is compatible with each budget. At inference time, the user specifies the budget b. The selected embedding $\mathbf{e}_b$ is concatenated with $\mathbf{z}$ to produce $\tilde{\mathbf{z}}$ with QoI and budget awareness.

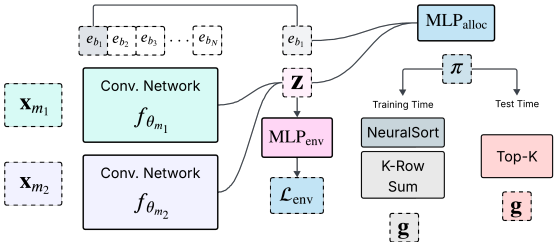

Fig. 2: `SWAN` controller. Lightweight convolutional networks extract the QoI of the input modalities, and *NeuralSort* maintains end-to-end differentiability during training

Finally, the controller selects an allocation of layers according to Eq. (2).

$$\boldsymbol{\pi} = \text{MLP}_{\text{alloc}}(\tilde{\mathbf{z}}) = [\boldsymbol{\pi}_1 \parallel \boldsymbol{\pi}_2 \parallel \cdots \parallel \boldsymbol{\pi}_M] \tag{2}$$

where each sub-vector $\boldsymbol{\pi}_m \in \mathbb{R}^{L_m}$ contains the logits used to determine the activation state of the L_m layers in the backbone of modality m. At inference time, we simply activate the layers corresponding to the b largest logits across $\boldsymbol{\pi}$. At training time, however, this sampling operation is non-differentiable and requires a different approach.

Training the Controller. The critical challenge when designing adaptive networks is maintaining differentiability during the training process. While methods such as Reinforcement Learning allow learning over non-differentiable operations, recent works [23,27] prefer Gumbel-Softmax [22] or Straight-Through [3] gradient approximation techniques for end-to-end learnability, citing the convergence struggles of reinforcement learning. However, we find that the straight-through propagation approach in ADMN results in uncertain layer selections that perform poorly on complex AV datasets (Tab. 1, Appendix).

In `SWAN`, we integrate the differentiable *NeuralSort* sorting algorithm into our controller to address this challenge [10]. Given a vector $\boldsymbol{\pi} \in \mathbb{R}^n$, NeuralSort produces a permutation matrix where the i-th row is defined as

$$\hat{P}_{sort(\boldsymbol{\pi})}[i,:](\tau) = \text{softmax}\left(\frac{(n+1-2i)\boldsymbol{\pi} - A_{\boldsymbol{\pi}}\mathbf{1}}{\tau}\right) \tag{3}$$

where $A_{\boldsymbol{\pi}}$ represents the matrix of absolute pairwise differences $|\boldsymbol{\pi}_i - \boldsymbol{\pi}_j|$ and τ is a temperature parameter that controls the smoothness of the relaxation. Intuitively, $\hat{P}_{sort(\boldsymbol{\pi})}[i,j]$ represents a soft assignment weight approximating the event that item j is the i-th largest element in $\boldsymbol{\pi}$.

During training, SWAN adds stochasticity through a standard Gumbel distribution to produce $\tilde{\boldsymbol{\pi}} = \text{Gumbel}(0,1) + \boldsymbol{\pi}$, and applies *NeuralSort* to obtain $\hat{P}_{sort(\tilde{\boldsymbol{\pi}})}$. With the previously sampled budget b, we calculate a soft-selection mask $\mathbf{g} \in \mathbb{R}^L$ (where $L = \sum_{m \in M} L_m$) by aggregating the first b rows of the relaxed permutation matrix:

$$\mathbf{g} = \sum_{i=1}^{b} \hat{P}_{\text{sort}(\tilde{\boldsymbol{\pi}})}[i,:] \tag{4}$$

To apply the budget constraint across modalities, we partition $\mathbf{g}$ into modality-specific sub-vectors $[\mathbf{g}_1 \parallel \mathbf{g}_2 \parallel \cdots \parallel \mathbf{g}_M]$. The forward pass for the l-th layer of modality m is then weighted by its corresponding gate $g_{m,l}$:

$$\mathbf{h}_{m,l} = g_{m,l} \cdot f_{m,l}(\mathbf{h}_{m,l-1}) + (1 - g_{m,l}) \cdot \mathbf{h}_{m,l-1} \tag{5}$$

where $\mathbf{h}_{m,l}$ is the output embedding of layer l and $f_{m,l}$ represents the current backbone layer.

Functionally, by annealing the temperature value τ during training, we can strike a balance between high *exploration* at the start of training, and *exploitation* during the latter stage. When τ is small (*e.g.* 0.1), the behavior of *NeuralSort* becomes near discrete, minimizing the domain shift between soft logits during training and hard-gating during inference. We supervise this process with the detection loss, allowing the model to learn layer allocations under various QoIs and compute budgets for ideal detection performance.

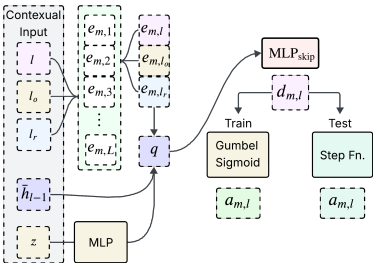

Fig. 3: SkipGate uses contextual information and the previous layer output for conditional execution

3.3 *SkipGate* Layer Dropout Module

SkipGate Architecture. While the controller effectively allocates b layers across all modalities, it will always consume the maximum allotted budget. Realistically, the user is likely to simply set a maximum compute budget that the system should not exceed. Consequently, adaptation within the layer budget can result in meaningful reductions in compute, energy, and latency.

In Fig. 3, we provide an overview of the proposed *SkipGate* module, which conditionally executes a layer selected by the QoI controller. We initialize individual *SkipGate* modules for each modality, and share the same module among

a modality's backbone layers. First, each SkipGate module maintains a fixed library of sinusoidal embeddings $\mathcal{E}_m = \{\mathbf{e}_{m,l} \in \mathbb{R}^d\}$ where $l \in \{0, 1 \dots L_{max}\}$ and L_{max} represents the maximum number of layers across all modality backbones. The SkipGate module converts the following contextual input integers into sinusoidal embeddings: the current layer in the backbone (l), the number of controller selected backbone layers remaining (l_r), and the number of layers assigned to the other modality (l_o). Additionally, the SkipGate module also receives the noise embeddings $\mathbf{z}$ from the controller. The need for all this contextual information underscores the difficulty of designing a SkipGate module in this setting – it is heavily influenced by the modality QoI and the decisions of the controller.

From this contextual information, SkipGate obtains logits $d_{m,l}$ representing the likelihood of executing a particular layer l according to Eq. (6).

$$\mathbf{q} = (\bar{h}_{l-1,m}) \parallel e_{m,l} \parallel \text{MLP}_z(\mathbf{z}) \parallel e_{m,l_r} \parallel e_{m,l_o} \quad ; \quad d_{m,l} = \text{MLP}_{\text{skip}}(\mathbf{q}) \tag{6}$$

where $\bar{h}_{l-1,m}$ is the averaged output features of a previous backbone layer. During inference time, we apply thresholding by executing only the layers with positive logit values.

SkipGate Training. Similar to the training of the QoI controller, layer selection is a non-differentiable operation. However, SkipGate only needs to make a single decision – whether to execute the current layer or not. Thus, we do not require the ordered property of NeuralSort and can resort to a simpler *Gumbel-Sigmoid* operator to obtain the soft-probabilities $a_{m,l}$:

$$a_{m,l} = \sigma\left(\frac{d_{m,l} + g_1 - g_2}{\tau}\right) \quad ; \quad g_1, g_2 \sim \text{Gumbel}(0,1) \tag{7}$$

As we decrease the temperature τ, the behavior approaches a discrete $(0, 1)$ selection while retaining differentiability. We then utilize $a_{m,l}$ to weigh the contributions of the current layer:

$$\mathbf{h}_{m,l} = a_{m,l} \cdot f_{m,l}(\mathbf{h}_{m,l-1}) + (1 - a_{m,l}) \cdot \mathbf{h}_{m,l-1} \tag{8}$$

We add a utilization penalty that encourages the module to minimize its layer usage, implemented as a *hinge loss* scaled by a constant β_m:

$$\mathcal{L}_{\text{skip}} = \sum_{m \in M} \sum_{l=1}^{L_m} \frac{\text{ReLU}(d_{m,l} + \beta)}{\beta_m} \tag{9}$$

$\mathcal{L}_{\text{skip}}$ serves two purposes – it penalizes large logit values, forcing the module to only activate the useful layers, and it cuts off the loss after $d_{m,l}$ reaches $-\beta$, preventing the module from "cheating" by making logits indefinitely negative.

3.4 DETR Head Token Pruning

The detection head converts a large number of modality tokens from the backbones into bounding box and classification results. With average LiDAR and

image resolutions, the detection head may process over thirty-thousand tokens. Despite this, many of the tokens may not be semantically meaningful in context of the downstream detection task, corresponding to irrelevant details such as the sky or background trees.

To remedy this, we propose a token dropping algorithm that eliminates these unnecessary tokens. Given the backbone output embeddings $\mathbf{h_m} \in \mathbb{R}^{H \times W \times D}$, we obtain a set of weights $\mathbf{w_m} \in \mathbb{R}^{H \times W}$, where $\mathbf{w_m} = \sigma(f_{\beta_m}(h_m))$. f_{β_m} is a small two-layer convolutional network that preserves the input shape through padding, and σ is the sigmoid function. We directly discretize $\mathbf{w_m}$ into $\tilde{\mathbf{w}}_\mathbf{m}$ by applying a rounding operation, and apply the straight through estimator for gradient propagation. The training process is supervised with an additional utilization loss $\mathcal{L}_{tp} = \sum_{m \in M} \sum_{i \in H, j \in W} (\tilde{w}_{m,i,j})$.

3.5 Joint Training Details

In this section, we provide greater details on the overall training process. *First*, we train our multimodal network with LayerDrop and standard detection loss $\mathcal{L}_{\text{detection}}$, producing a layer-adaptive multimodal AV network. *Next*, we freeze the main network's parameters and train the *QoI-aware controller* on the corrupted nuScenes dataset with loss $\mathcal{L} = \mathcal{L}_{\text{detection}} + \alpha_1 \mathcal{L}_{\text{env}}$, where $\mathcal{L}_{\text{env}}$ is from Eq. (1). Afterwards, we introduce the *SkipGate* modules while freezing all other parameters, and train with loss $\mathcal{L} = \mathcal{L}_{\text{detection}} + \alpha_2 \mathcal{L}_{\text{skip}}$, where $\mathcal{L}_{\text{skip}}$ is defined in Eq. (9). We anneal the temperature τ during training. Finally, we train for token pruning with loss $\mathcal{L} = \mathcal{L}_{\text{detection}} + \alpha_3 \mathcal{L}_{\text{tp}}$. α_2 and α_3 allow for control over the degree of layer skipping and token pruning by trading off higher accuracy with greater efficiency. Further details are in the Appendix.

4 Results

4.1 Main MultiCorrupt Results

Datasets and Baselines. The standard nuScenes dataset [5] contains only a limited number of rainy and dark corruptions. Since SWAN targets varying modality QoI, we used the *MultiCorrupt* [2] generation pipeline to simulate five new conditions by corrupting only the clean (sunny, dry) samples of the nuScenes dataset. *Lidar Beamsreducing* occurs when sensor degradation or power constraints restrict the number of emitted beams, *Camera Fog* simulates a foggy scene where camera is heavily impacted while high-power LiDARs exhibit robustness, *Darkness* heavily impacts camera while LiDAR is unaffected, *LiDAR Motionblur* and *Camera Motionblur* occur when vehicle vibrations or improper mounting lead to motion artifacts. We emphasize that these evaluations were merely a conduit to display SWAN's robustness towards dynamic QoI – we did not aim to exhaustively evaluate every possible environmental corruption.

We compared our base AV network against several popular AV detection models: LiDAR-only SST [9], image-only PETR [18], and multimodal BEVFusion [20]. Moreover, to simulate platform dynamics, we apply SWAN under different budgets of L total backbone layers. Under this constraint, we implement two

Table 1: `SWAN` against baselines across diverse modality QoI and compute budgets. The three SWAN variants progressively integrate the controller (C), SkipGates (S), and token pruner (P). Metrics reported are NDS and mAP with averaged median latency (ms) and Giga-Floating Point Operations (GFLOPs) on an Nvidia RTX 4090

Method	LiDAR Beamsreduce		Camera Fog		Camera Motionblur		Dark		LiDAR Motionblur		Compute ↓	
	NDS ↑	mAP ↑	NDS	mAP	NDS	mAP	NDS	mAP	NDS	mAP	GFLOPs	Latency
Base Network	41.80	28.97	67.57	61.30	67.65	61.31	66.85	59.45	63.93	58.32	651.00	124.84
BEVFusion	32.08	10.77	68.38	62.89	69.49	64.88	68.28	62.56	58.18	53.60	630.73	118.76
PETR	35.21	32.63	22.17	14.72	20.32	12.84	23.24	14.92	35.21	32.63	1100.15	44.31
SST	22.06	1.38	66.42	58.65	66.43	58.65	66.43	58.65	54.82	42.73	372.85	92.79
Naive 16	39.48	25.40	67.48	61.17	67.21	60.85	66.70	59.37	63.02	56.79	556.41	120.93
SWAN-C 16	40.97	27.74	67.31	60.62	67.31	60.55	67.2	60.49	63.25	57.3	597.55	121.59
SWAN-SC 16	39.01	28.37	66.18	58.16	66.18	58.46	65.86	58.48	63.03	56.94	455.66	116.17
SWAN-PSC 16	36.42	25.43	65.08	57.43	64.95	57.20	64.72	57.34	61.40	55.18	428.37	83.42
ADMN 16	40.12	26.78	67.54	61.16	67.28	61.15	66.89	60.15	62.84	56.65	584.01	123.21
Naive 8	32.92	16.16	64.88	56.74	64.39	56.34	64.74	56.47	59.33	50.97	426.56	106.09
SWAN-C 8	36.76	26.08	64.57	56.67	64.83	56.81	65.92	57.83	58.37	49.21	445.01	110.33
SWAN-SC 8	36.77	26.07	64.56	55.95	65.12	57.02	66.05	58.33	58.35	49.13	400.63	109.03
SWAN-PSC 8	34.42	23.23	63.99	55.95	64.20	56.24	64.94	57.14	56.82	47.60	373.42	77.15
ADMN 8	36.35	21.04	65.86	57.90	65.69	57.79	65.68	57.74	55.74	46.52	438.09	110.21
Naive 6	29.59	9.99	61.40	51.02	61.73	52.10	61.34	50.74	54.78	44.17	394.31	102.11
SWAN-C 6	34.33	22.61	63.77	54.72	64.55	55.95	64.28	55.80	56.19	45.18	400.43	107.09
SWAN-SC 6	34.30	22.60	64.45	55.78	65.24	57.20	65.50	57.58	54.41	41.98	372.62	106.51
SWAN-PSC 6	31.99	19.87	63.96	55.92	64.36	56.47	64.52	56.66	55.00	44.08	345.07	74.37
ADMN 6	32.55	15.95	63.06	53.71	62.94	53.82	63.12	53.99	48.05	35.29	407.78	105.81
Naive 4	24.71	3.88	57.83	44.70	57.71	45.02	56.73	43.62	45.82	29.45	362.00	98.81
SWAN-C 4	29.17	16.12	64.47	55.80	64.77	56.49	64.46	56.13	54.38	41.97	358.90	103.73
SWAN-SC 4	29.16	16.11	64.45	55.78	64.72	56.48	64.45	56.10	54.33	41.94	358.90	104.33
SWAN-PSC 4	27.92	14.18	63.95	55.90	64.95	55.91	63.74	55.69	54.86	43.99	331.41	72.21
ADMN 4	26.33	7.48	55.69	42.04	60.55	49.66	58.08	46.47	47.82	32.20	369.65	102.02

additional baselines – Naive Allocation where each modality receives $\frac{L}{2}$ layers, and also the controller-based, QoI-aware ADMN [27] scheme. For greater transparency into `SWAN`, we progressively evaluate the integration of each module, from the controller to token pruning. Unless otherwise specified, `SWAN` refers to SWAN-PSC from Tab. 1.

Key Takeaways. We summarize the key observations from Tab. 1, followed by a more comprehensive analysis in the subsequent sections. First, `SWAN`'s controller-based allocation (SWAN-C) substantially increases mAP and NDS compared to similar baselines under compute-limited settings (*e.g.* four layers). Next, the SkipGate (SWAN-SC) module's ability to drop layers reduces the model FLOPs by up to 18% below the Naive baseline with marginal decreases in mAP and NDS. Finally, the token pruning module (SWAN-PSC) filters out a large portion of tokens prior to the detection head, reducing the SWAN-SC latency by a maximum of 30.8%.

SWAN Detection Performance. In the first row of Tab. 1, we compare the base multimodal AV detection network with all 20 layers (8 in LiDAR, 12 in image) against multimodal (BEVFusion), LiDAR-only (SST), and camera-only (PETR) networks. Our multimodal network displays enhanced robustness

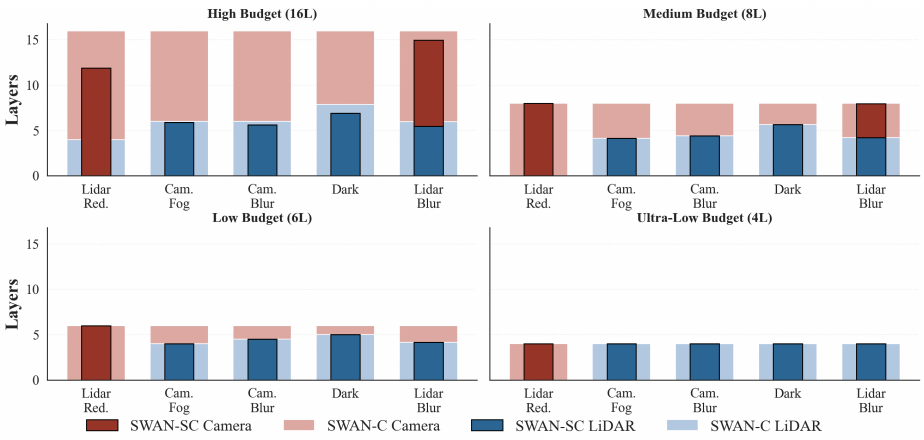

Fig. 4: Layer selection of SWAN-C and SWAN-SC under different corruptions and budgets. Maximum budget is 20 layers

against environmental corruptions compared to the unimodal networks: when either LiDAR in SST or camera in PETR is corrupted, the lack of redundancy causes model performance to drop. Additionally, our base multimodal network is on-par with the efficient BEVFusion's performance.

Applying SWAN under compute restricted regimes results in meaningful increases in detection performance. For instance, under 4 layers of budget and LiDAR Beamsreduce noise, adding the controller (SWAN-C) raises mAP from 3.88 to 16.12 compared to the Naive baseline. Similarly large NDS and mAP gains occur across other corruptions. Moreover, SWAN consistently outperforms ADMN under budget scarcity. As shown in the Appendix, ADMN's straight-through gradient estimation fails to learn precise layer allocations – it selects suboptimal layers within each modality's backbone (apparent under LiDAR motionblur). In contrast, SWAN utilizes *NeuralSort* for controller training, enabling a superior understanding of relative backbone layer importance which directly translates into improved detection performance.

Additionally, when comparing against the fully-provisioned *Base Network*, SWAN achieves competitive results with a fraction of the required layers. With a 4 layer budget, SWAN detects within 2.7 NDS and 5.4 mAP of this upper bound under Camera Motionblur. Similar results can be observed under Darkness. This highlights not only the effectiveness of our layer allocation algorithm, but also the flexiblity of our multimodal adaptive backbone.

Controller and SkipGate Layer Selection. Next, we take a deeper dive into the layer allocations outputted by the SkipGate module and controller. Tab. 1 shows how the SkipGate module drastically reduces the compute performed in the network when the allocated budget is high without an adverse impact on detection performance. For instance, on average, SWAN-SC reduces GFLOPs by over 18% relative to the Naive baseline at a budget of 16 Layers.

Fig. 4 outlines the layer selection capabilities of the controller (SWAN-C) and SkipGate (SWAN-SC) modules. Under severe budget constraints (*e.g.* 4 layers), the controller favors modalities with higher relative QoI (*e.g.* LiDAR during darkness). Interestingly, under LiDAR motionblur, the controller still favors the LiDAR modality. As showcased in Tab. 1, diverting resources towards camera (*i.e.* Naive 4) results in significant degradation, demonstrating that the blurry LiDAR still offers superior information compared to camera. This reveals one of the key advantages of `SWAN`: rather than relying on pre-conceived human notions of how to allocate resources, `SWAN` *learns an optimal allocation from the data itself*. Furthermore, in these low budgets, the SkipGate module recognizes the resource scarcity and correctly decides to executes all layers.

The utility of the SkipGate module becomes evident under compute-rich settings (*e.g.* 16 layers). In this setting, there exists redundancy across layers, allowing the SkipGate module to eliminate certain layers to reduce computation. The SkipGate module analyzes the input to determine the layer execution – running as few as 6 layers (*e.g.* camera fog) or as many as 15 layers (*e.g.* LiDAR Blur), demonstrating its advanced understanding of how the current budget and environmental conditions affect the necessity of a given layer.

Token Pruning. We calculate that on average, SWAN-PSC retains 25.54% of image tokens and 48.84% of LiDAR tokens, contributing towards large decreases in model latency in Tab. 1. Fig. 5 depicts how the pruning module prioritizes semantically meaningful areas by removing the background LiDAR and image features. Moreover, when camera is corrupted by fog, most of the image tokens are discarded, and the few retained tokens surround the detection targets (*e.g.* vehicles) in the scene. However, we emphasize that these visualizations are an approximation. Due to downsampling and windowed attention in both transformer backbones, information is exchanged and condensed across spatial tokens, which can result in the irregular pattern seen in "Clean Camera".

Latency and FLOPs Analysis. Interestingly, despite the SkipGate module reducing layer utilization (Fig. 4) and minimizing FLOPs by up to 18%, the latency barely improves over the Naive 16 baseline (Tab. 1). Additionally, under other layer budgets, SWAN-SC can take longer to execute compared to both the Naive baseline and SWAN-C. We find that on average, the controller takes 1.5 ms to run while each SkipGate image and LiDAR module have latencies of 0.2 and 0.3 ms, respectively, failing to account for the discrepancy in the results.

We attribute this "invisible latency" to the *orchestration overhead of PyTorch* on high-performance devices. The controller and SkipGate modules require data (*e.g.* budgets, allocation results) to be transferred between the CPU and GPU, which can take fractions of a millisecond. On a high performance system (*e.g.* Nvidia RTX 4090) where the backbone layers can run in milliseconds, the orchestration latency becomes a non-negligible portion of model runtime. We verify this in Section 4.3, where an Nvidia Jetson Orin edge device running the same `SWAN` models benefits by over 100 ms with the inclusion of the SkipGate modules. Additionally, we discuss model compilation to mask this latency in Section 5.

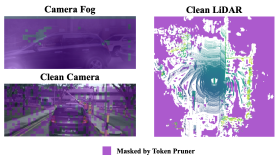

Fig. 5: Token Pruning Visualization

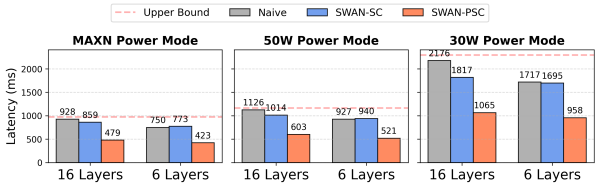

Fig. 6: Evaluations on Nvidia Jetson Orin across camera fog corruption and different power modes

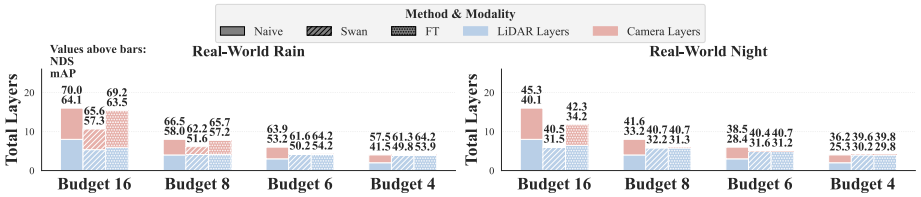

Fig. 7: Base and Finetuned (FT) SWAN on real-world nuScenes rainy/dark data. *Bar height represents layer utilization, with NDS/mAP scores shown above in text*

4.2 Real Corrupted Data

To showcase the feasibility of SWAN on real-world noisy data, we validate it on rainy and dark subsets of the nuScenes validation dataset. We previously discarded this data to avoid interference with MultiCorrupt data synthesis. First, we perform a challenging evaluation of *sim2real transfer coupled with novel corruption generalization*. SWAN has only seen MultiCorrupt dark samples, and has never seen any rainy samples during the training of its modules. In Fig. 7, we showcase SWAN's performance compared to the Naive (equal) allocation baseline. Despite the domain shift, SWAN obtains superior NDS/mAP in low-compute regimes, confirming the generality of our approach.

However, SWAN suffers from low accuracy on higher budget allocations due to extreme layer dropping by the SkipGate module. We utilized heavily corrupted synthetic data to train SWAN, which likely resulted in the SkipGate model learning aggressive layer dropping. Thus, we independently finetune the SkipGate module and the subsequent token pruning on the rainy and dark subset of nuScenes train. Fig. 7 depicts higher layer usage and recovered NDS/mAP.

The need to perform real-world finetuning to recover accuracy is not a shortcoming of SWAN – it stems from the domain shift between the synthetic Multicorrupt dataset and real-world nuScenes samples. When available, SWAN should always be trained and tested on real-world corrupted data to minimize the domain gap. Existing AV datasets prioritize high-quality sensing and lack the systematic modality failures (e.g., severe sensor faults) to fully evaluate SWAN's capabilities, necessitating the usage of the Multicorrupt dataset.

Table 2: MM-Fi 15-Frame Human Gesture Recognition evaluated on RTX 4090. We showcase classification accuracy (%) ↑ and PyTorch latency (ms) ↓ with percent overhead (Δ%) with respect to Naive Alloc latency. Format: Accuracy/Latency (Δ%).

Method	6 Layers	10 Layers	16 Layers	20 Layers
Naive	23.2/12.7	33.0/**18.2**	**45.7**/26.3	44.4/31.9
SWAN-C	**30.9**/13.7(+7.4)	**39.2**/19.3(+6.1)	43.2/27.5(+4.7)	**46.0**/33.1(+3.6)
SWAN-SC	29.6/**12.3**(-3.3)	38.0/18.7(+2.9)	42.9/**24.0**(-8.5)	41.7/**27.1**(-15.1)

4.3 Edge Hardware Evaluation

We benchmark our SWAN model on an Nvidia Jetson Orin edge device on camera fog corruption in Fig. 6. We observe a substantial reduction in per-sample latency when introducing the SkipGate module (SWAN-SC), reinforcing our hypothesis that the Tab. 1 latencies on the RTX 4090 were dominated by overhead costs. Moreover, as the hardware capabilities degrade, the advantage of the SkipGate controller increases, reducing the latency compared to the Naive baseline by 7.4%, 9.9%, and 16.5% for MAXN, 50W, and 30W power constraints, respectively. When employing the full SWAN model with token pruning, we reduce latency by over 50%. Real-world AVs are likely to utilize these power-efficient edge devices, highlighting SWAN's impact in realistic deployments.

4.4 Broader Evaluations

We showcase that SWAN generalizes beyond autonomous vehicles with LiDAR and RGB camera to general multimodal tasks by evaluating on the multimodal gesture recognition *MM-Fi* [29] dataset. We utilize the *depth camera and RGB camera* modalities and add synthetic Gaussian Noise corruption. We pass 15 frames of data to the modality-specific ViT-Base encoder for each sample, then aggregate across modalities and time with a transformer encoder. Consistent with the results on nuScenes, Tab. 2 shows how SWAN improves performance at lower budgets while reducing latency through sample-aware layer dropping.

5 Discussion and Limitations

SWAN Applicability. Although we primarily evaluated on the nuScenes dataset, we reiterate that SWAN's contributions are equally applicable to not only other types of AVs (*e.g.* mobile robots), but also in any resource constrained environment with QoI variations (*e.g.* wearables, surveillance).

Unit of Resource Abstraction. To focus on quality-aware resource allocation, we utilize layer count as a primary unit of budget abstraction. While LiDAR and Image layers are not equally resource intensive – image layers consume ∼2.4x more FLOPs but ∼1/3 the latency of LiDAR layers – this abstraction enables a clear evaluation of the controller's ability to prioritize modalities based on

marginal utility. Consequently, the FLOPs and latency values in Tab. 1 are slightly influenced by the reallocation between these asymmetric layers. However, our core contributions surrounding the differentiable NeuralSort-based controller, input-adaptive SkipGate, and token pruning modules are agnostic to the specific cost metric used, and the significant gains in robustness and efficiency in Tab. 1 remain the primary takeaway. Future extensions could incorporate non-uniform "layer costs" into the controller's loss function, as demonstrated in ADMN [27], without altering SWAN's underlying architecture.

Model Compilation. When analyzing the latency of Tab. 1, we remarked on the *orchestration overhead* that diminished SWAN's latency reductions. Future work can explore model compilation frameworks such as TensorRT [24], which fuse individual operations into a complete GPU graph to minimize launch overhead of small kernels. TensorRT also provides support for GPU-side conditionals, avoiding costly inter-device data transfer and CPU orchestration.

6 Conclusion

We present SWAN, the first multimodal neural network jointly addressing runtime variations across modality QoI, platform dynamics, and input complexity. Its QoI-aware controller assigns resources to modality backbones under a strict compute budget, while the input-aware SkipGate and token pruning modules enable further efficiency. Evaluations on a corrupted variant of the nuScenes dataset demonstrates that SWAN achieves comparable accuracy to a fully-provisioned network despite utilizing only a fraction of the resources. SWAN also retains performance on real-world data with exciting results on edge deployments.

7 Acknowledgements

The research reported in this paper was sponsored in part by the U.S. Army DEVCOM Army Research Laboratory (award #W911NF1720196) and the National Science Foundation (awards #CNS-2211301 and CNS-2325956). The Department of Defense (DoD) supported Jason Wu through the National Defense Science & Engineering Graduate (NDSEG) Fellowship Program. This material is based upon work supported by the Air Force Office of Scientific Research under award number FA9550-23-F-0014 in the amount of $36,000. The views and conclusions contained in this document are those of the author(s) and should not be interpreted as representing the official policies of the funding agencies.

References

1. Bai, X., Hu, Z., Zhu, X., Huang, Q., Chen, Y., Fu, H., Tai, C.L.: Transfusion: Robust lidar-camera fusion for 3d object detection with transformers (2022), https://arxiv.org/abs/2203.11496

2. Beemelmanns, T., Zhang, Q., Geller, C., Eckstein, L.: Multicorrupt: A multi-modal robustness dataset and benchmark of lidar-camera fusion for 3d object detection. In: 2024 IEEE Intelligent Vehicles Symposium (IV). pp. 3255–3261. IEEE (2024)
3. Bengio, Y., Léonard, N., Courville, A.: Estimating or propagating gradients through stochastic neurons for conditional computation. arXiv preprint arXiv:1308.3432 (2013)
4. Bolukbasi, T., Wang, J., Dekel, O., Saligrama, V.: Adaptive neural networks for fast test-time prediction. arXiv preprint arXiv:1702.07811 **1**(3) (2017)
5. Caesar, H., Bankiti, V., Lang, A.H., Vora, S., Liong, V.E., Xu, Q., Krishnan, A., Pan, Y., Baldan, G., Beijbom, O.: nuscenes: A multimodal dataset for autonomous driving. In: Proceedings of the IEEE/CVF conference on computer vision and pattern recognition. pp. 11621–11631 (2020)
6. Cai, H., Gan, C., Wang, T., Zhang, Z., Han, S.: Once-for-all: Train one network and specialize it for efficient deployment. arXiv preprint arXiv:1908.09791 (2019)
7. Chen, X., Zhang, T., Wang, Y., Wang, Y., Zhao, H.: Futr3d: A unified sensor fusion framework for 3d detection. In: proceedings of the IEEE/CVF conference on computer vision and pattern recognition. pp. 172–181 (2023)
8. Fan, A., Grave, E., Joulin, A.: Reducing transformer depth on demand with structured dropout. arXiv preprint arXiv:1909.11556 (2019)
9. Fan, L., Pang, Z., Zhang, T., Wang, Y.X., Zhao, H., Wang, F., Wang, N., Zhang, Z.: Embracing single stride 3d object detector with sparse transformer. In: Proceedings of the IEEE/CVF conference on computer vision and pattern recognition. pp. 8458–8468 (2022)
10. Grover, A., Wang, E., Zweig, A., Ermon, S.: Stochastic optimization of sorting networks via continuous relaxations. arXiv preprint arXiv:1903.08850 (2019)
11. Hu, B., Xu, L., Moon, J., Yadwadkar, N.J., Akella, A.: Mosel: Inference serving using dynamic modality selection (2023), https://arxiv.org/abs/2310.18481
12. Huang, G., Chen, D., Li, T., Wu, F., Van Der Maaten, L., Weinberger, K.Q.: Multi-scale dense networks for resource efficient image classification. arXiv preprint arXiv:1703.09844 (2017)
13. Ji, M., Zhang, S., Yang, J.: Depthfusion: Depth-aware hybrid feature fusion for lidar-camera 3d object detection. IEEE Transactions on Multimedia (2026)
14. Jing, W., Chen, X., Nie, S., Jiao, Z., Ma, J.: Hd-fusion: Hierarchical dynamic fusion of lidar-camera for robust 3-d object detection. IEEE Transactions on Industrial Informatics (2026)
15. Li, H., Zhang, H., Qi, X., Yang, R., Huang, G.: Improved techniques for training adaptive deep networks. In: Proceedings of the IEEE/CVF international conference on computer vision. pp. 1891–1900 (2019)
16. Li, Y., Lyu, C., Wang, H.: Freqexit: Enabling early-exit inference for visual autoregressive models via frequency-aware guidance. In: Belgrave, D., Zhang, C., Lin, H., Pascanu, R., Koniusz, P., Ghassemi, M., Chen, N. (eds.) Advances in Neural Information Processing Systems. vol. 38, pp. 103763–103788. Curran Associates, Inc. (2025), https://proceedings.neurips.cc/paper_files/paper/2025/file/95cb60b65d9f2c0cd78b567ffebde9e6-Paper-Conference.pdf
17. Liu, L., Su, B., Jiang, J., Wu, G., Guo, C., Xu, C., Yang, H.F.: Towards accurate and efficient 3d object detection for autonomous driving: A mixture of experts computing system on edge. In: Proceedings of the IEEE/CVF International Conference on Computer Vision. pp. 25903–25913 (2025)
18. Liu, Y., Wang, T., Zhang, X., Sun, J.: Petr: Position embedding transformation for multi-view 3d object detection. In: European conference on computer vision. pp. 531–548. Springer (2022)

19. Liu, Z., Lin, Y., Cao, Y., Hu, H., Wei, Y., Zhang, Z., Lin, S., Guo, B.: Swin transformer: Hierarchical vision transformer using shifted windows. In: Proceedings of the IEEE/CVF international conference on computer vision. pp. 10012–10022 (2021)
20. Liu, Z., Tang, H., Amini, A., Yang, X., Mao, H., Rus, D.L., Han, S.: Bevfusion: Multi-task multi-sensor fusion with unified bird's-eye view representation. In: 2023 IEEE international conference on robotics and automation (ICRA). pp. 2774–2781. IEEE (2023)
21. Liu, Z., Yang, X., Tang, H., Yang, S., Han, S.: Flatformer: Flattened window attention for efficient point cloud transformer. In: Proceedings of the IEEE/CVF conference on computer vision and pattern recognition. pp. 1200–1211 (2023)
22. Maddison, C.J., Mnih, A., Teh, Y.W.: The concrete distribution: A continuous relaxation of discrete random variables. arXiv preprint arXiv:1611.00712 (2016)
23. Meng, L., Li, H., Chen, B.C., Lan, S., Wu, Z., Jiang, Y.G., Lim, S.N.: Adavit: Adaptive vision transformers for efficient image recognition. In: Proceedings of the IEEE/CVF conference on computer vision and pattern recognition. pp. 12309–12318 (2022)
24. NVIDIA Corporation: NVIDIA TensorRT: High-performance deep learning inference sdk (2024), https://developer.nvidia.com/tensorrt, accessed: 2026-03-04
25. Panda, R., Chen, C.F.R., Fan, Q., Sun, X., Saenko, K., Oliva, A., Feris, R.: Adamml: Adaptive multi-modal learning for efficient video recognition. In: Proceedings of the IEEE/CVF international conference on computer vision. pp. 7576–7585 (2021)
26. Rao, Y., Zhao, W., Liu, B., Lu, J., Zhou, J., Hsieh, C.J.: Dynamicvit: Efficient vision transformers with dynamic token sparsification. Advances in neural information processing systems **34**, 13937–13949 (2021)
27. Wu, J., Yuan, Y., Yang, K., Kaplan, L., Srivastava, M.: Admn: A layer-wise adaptive multimodal network for dynamic input noise and compute resources (2025), https://arxiv.org/abs/2502.07862
28. Yan, J., Liu, Y., Sun, J., Jia, F., Li, S., Wang, T., Zhang, X.: Cross modal transformer: Towards fast and robust 3d object detection. In: Proceedings of the IEEE/CVF international conference on computer vision. pp. 18268–18278 (2023)
29. Yang, J., Huang, H., Zhou, Y., Chen, X., Xu, Y., Yuan, S., Zou, H., Lu, C.X., Xie, L.: Mm-fi: Multi-modal non-intrusive 4d human dataset for versatile wireless sensing. In: Thirty-seventh Conference on Neural Information Processing Systems Datasets and Benchmarks Track (2023), https://openreview.net/forum?id=1uAsASS1th