AdaptViG: Adaptive Vision GNN with Exponential Decay Gating

Supplementary Material

A. Theoretical Basis of Adaptive Graph Convolution

A.1. Foundations of Graph-Based Information Aggregation

In vision models, a feature map can be viewed as a grid graph where each pixel (or patch embedding) is a node. A graph convolution operation aims to update a node's feature vector by aggregating information from its neighbors. Let $X \in \mathbb{R}^{B\times C\times H\times W}$ be an input feature map. For a node v_p at position p with feature vector $x_p \in \mathbb{R}^C$, a general graph update rule can be written as:

$$x'_p = \phi\left(x_p, \bigoplus_{q\in\mathcal{N}(p)} \psi(x_p, x_q)\right) \tag{7}$$

where $\mathcal{N}(p)$ is the set of neighbors of node p, ψ is a message function computing the relationship between nodes, $\bigoplus$ is an aggregation function (e.g., sum, mean, max), and ϕ is an update function (e.g., an MLP).

Our AGC (Adaptive Graph Convolution) module implements a specific form of this update. We define the aggregated information from all neighbors as a single "max-relative feature" tensor $X_j \in \mathbb{R}^{B\times C\times H\times W}$. The message function ψ is the feature difference $(x_q - x_p)$, and the aggregation $\bigoplus$ is the element-wise maximum. The final update ϕ is a concatenation followed by a convolutional block.

A.2. The Static Scaffold: An Efficient Structural Prior

The foundation of our AGC module is a highly efficient, predefined graph structure we term the **static scaffold**. This scaffold defines the neighborhood set $\mathcal{N}(p)$ for every node p and is designed to capture a rich set of spatial relationships without relying on computationally expensive search algorithms like KNN. The scaffold is composed of two distinct types of connections, ensuring a comprehensive receptive field. The aggregated feature map for the scaffold, $X_{j,\text{static}}$, is initialized as a zero tensor.

1. Fixed Local Connections To ensure a strong inductive bias for local feature interactions, we establish connections to neighbors at a fixed distance, K. This is similar to a dilated convolution and is crucial for capturing fine-grained details. Let roll(X, s, a) be an operator that cyclically shifts tensor X by s positions along axis a. The update rule uses one shift per spatial axis:

$$X_j \leftarrow \max(X_j, \text{roll}(X, -K, H) - X) \tag{8}$$
$$X_j \leftarrow \max(X_j, \text{roll}(X, -K, W) - X) \tag{9}$$

where H and W correspond to height and width axes where the roll operations occur.

2. Logarithmic Long-Range Connections To model the global context efficiently, we form connections to neighbors at exponentially increasing distances. This allows information to propagate across the entire feature map in a logarithmic number of steps, rather than linear. For each spatial dimension, we iterate and update X_j:

$$\forall i \in [1, \lfloor\log_2 H\rfloor] : X_j \leftarrow \max(X_j, \text{roll}(X, -2^i, 2) - X) \tag{10}$$
$$\forall i \in [1, \lfloor\log_2 W\rfloor] : X_j \leftarrow \max(X_j, \text{roll}(X, -2^i, 3) - X) \tag{11}$$

The combination of these two strategies creates the static scaffold, providing a robust structural prior that captures both short- and long-range dependencies with minimal computational overhead.

A.3. Exponential Decay Gating for Content-Awareness

While the static scaffold is efficient, it is not content-aware. To address this, we introduce a *dynamic gating* mechanism that weighs connections based on semantic similarity. The core of this mechanism is the **Exponential Decay Gate**, a function designed for numerical stability.

For a given patch, $x_p \in \mathbb{R}^C$ and a potential neighbor x_n, we define a gate value $g_{pn} \in (0, 1]$ that modulates their connection.

Definition A.1: Feature Distance. We first compute the dissimilarity between the two patches using the L1 norm, which is computationally efficient and robust to outliers.

$$d(x_p, x_n) = ||x_p - x_n||_1 = \sum_{c=1}^{C} |x_{p,c} - x_{n,c}| \tag{12}$$

Definition A.2: Exponential Decay Gate. The distance $d(x_p, x_n)$ is then transformed into a gating value using an exponential decay function.

$$g(x_p, x_n) = \exp\left(-\frac{d(x_p, x_n)}{T}\right) \tag{13}$$

Here, T is a learnable scalar parameter (**temperature**).

Rationale and Numerical Stability. A key motivation for this gating function is to avoid numerical instability, which often leads to ‘NaN‘ losses during training.

- **No Division by Features:** The function in Equation 13 does not divide by any value derived from the input features. This is critical as it ensures there will be no divide by zero cases even if the input has zero variance (e.g., a uniform patch in an image).
- **Bounded Output:** The output of the exponential gate is naturally bounded. Since the L1 distance is always non-negative ($d \geq 0$) and we enforce $T > 0$, the argument to ‘exp‘ is always non-positive ($-\frac{d}{T} \leq 0$). This guarantees the output is bounded within the range $(0, 1]$ and cannot explode to infinity.
- **Learnable Sensitivity:** The temperature T provides crucial flexibility. We constrain it to be positive by using $|T| + \epsilon$ in the implementation. The model learns the optimal sensitivity for each layer.

A.4. Gradient Flow Analysis

To understand how the network learns to form meaningful connections, we discuss the gradient flow from the loss function $\mathcal{L}$ back to the parameters of the gating mechanism, specifically the temperature T.

Let us consider the contribution of a single gated connection to the aggregated feature X_j. Let the message from a neighbor be $M_{pn} = x_n - x_p$. The gated message is $M'_{pn} = g_{pn} \odot M_{pn}$ where $\odot$ is the element-wise product. The update to X_j at position p involves this term. For simplicity, let's assume the loss $\mathcal{L}$ is a function of X_j. The gradient of the loss with respect to the gate value g_{pn} is given by the chain rule:

$$\frac{\partial \mathcal{L}}{\partial g_{pn}} = \frac{\partial \mathcal{L}}{\partial X_j(p)} \frac{\partial X_j(p)}{\partial M'_{pn}} \frac{\partial M'_{pn}}{\partial g_{pn}} \tag{14}$$

Since $\frac{\partial M'_{pn}}{\partial g_{pn}} = M_{pn} = (x_n - x_p)$, this becomes:

$$\frac{\partial \mathcal{L}}{\partial g_{pn}} = \nabla_{X_j(p)} \mathcal{L} \cdot (x_n - x_p) \tag{15}$$

This shows that the gradient for a gate depends on how much the feature difference $(x_n - x_p)$ aligns with the overall gradient of the loss. If incorporating the neighbor's information helps reduce the loss, the gradient will push the gate value g_{pn} higher, strengthening the connection.

Now, let's analyze the gradient for the temperature T, which controls the gate's sensitivity. Using the chain rule again:

$$\frac{\partial \mathcal{L}}{\partial T} = \sum_{p,n} \frac{\partial \mathcal{L}}{\partial g_{pn}} \frac{\partial g_{pn}}{\partial T} \tag{16}$$

The derivative of the gate function with respect to T is:

$$\frac{\partial g_{pn}}{\partial T} = \frac{\partial}{\partial T} \exp\left(\frac{-d_{pn}}{T}\right) = \exp\left(\frac{-d_{pn}}{T}\right) \cdot \frac{d_{pn}}{T^2} = g_{pn} \frac{d_{pn}}{T^2} \tag{17}$$

Substituting this back in Equation 16, we get:

$$\frac{\partial \mathcal{L}}{\partial T} = \sum_{p,n} \left(\nabla_{X_j(p)} \mathcal{L} \cdot (x_n - x_p)\right) \left(g_{pn} \frac{d_{pn}}{T^2}\right) \tag{18}$$

The update rule for T during optimization will be $T \leftarrow T - \eta \frac{\partial \mathcal{L}}{\partial T}$. Equation (18) reveals that the learning signal for T is proportional to the distance d_{pn}. This means the model learns a single temperature T that best balances the trade-off across all connections: it will be pushed to a value that appropriately suppresses connections with large, unhelpful distances while preserving connections with small, useful distances, all in service of minimizing the global loss $\mathcal{L}$. This provides a strong theoretical justification for the mechanism's ability to learn meaningful graph structures.

B. Additional Ablation Studies

We conduct ablation studies on ImageNet-1K [11] to validate the key design choices of our AdaptViG architecture. Unless otherwise specified, the ablations are performed on the AdaptViG-B model.

B.1. Static Connection Distance (K).

The distance K for the guaranteed local connections in the static scaffold of our AGC blocks can affect performance. We test various configurations for the K values used in Stages 1, 2, and 3. As shown in Table 5, we find that using K values of [8, 4, 2] for these three stages, respectively, yields the best performance. Increasing the initial hop distance (K = [16, 8, 4]) results in a 0.1% accuracy drop, while a different pattern (K = [9, 6, 3]) leads to a 0.1% drop, validating our chosen configuration.

Table 5. **Ablation on the static connection distance (K)** for the AGC blocks in AdaptViG-B. The Top-1 accuracy results are averaged over two experiments.

K (for AGC Stages 1-3)	Params (M)	Top-1 (%)
K = 16, 8, 4	26.8	83.2
K = 9, 6, 3	26.8	83.2
K = 8, 4, 2	**26.8**	**83.3**

B.2. Computational Complexity

The computational complexity of the graph construction step is a critical factor in the efficiency of Vision GNNs. Here, we analyze the per-node complexity for different methods. Let W and H be the width and height of the feature map, K be the number of nearest neighbors for KNN, and N is the number of hops selected in SVGA and DAGC.

1. **KNN** [9, 15]: O($W \times H$). For each node, KNN must compare against all other nodes in the feature map, making it computationally prohibitive.

2. **DAGC** [42]: O($\frac{W+H}{N}$). This method connects nodes at linear intervals with a stride of N, making its complexity dependent on the feature map's dimensions divided by the hop distance.
3. **AGC (Ours)**: O($\log W + \log H$). Our method connects nodes at logarithmic intervals, making it highly efficient and scalable while still capturing global, content-aware relationships.
4. **SVGA** [41]: O(1). As a static method, the graph is fixed, and the connections are retrieved via efficient shift operations with constant time complexity per node.

Our AGC is marginally more expensive than a purely static method like SVGA, but is more efficient than both KNN and DAGC. To demonstrate this, we measure the graph construction time of a single forward pass on an Nvidia RTX A6000. We measure an average time of 0.048 ms for our AGC, which is 20% faster than DAGC (0.06 ms) and significantly faster than KNN (0.38 ms), while being nearly on par with the static SVGA (0.04 ms).

This efficiency translates into better end-to-end model latency per batch, as shown in Table 6. Our **AdaptViG-S w/ AGC** is significantly faster than a comparable model using KNN and is also more accurate and faster than PViG-Ti using KNN. This highlights the effectiveness of our architecture and the efficiency of the AGC module.

Table 6. **Graph construction impact on accuracy and latency.** The table shows that our AGC provides a better accuracy-latency trade-off compared to KNN, DAGC, and static SVGA methods for models of similar size.

Model	Params	Latency	Acc (%)
MobileViG-S [41] w/ SVGA	7.2 M	27.1 ms	78.2
PViG-Ti [15] w/ KNN	10.7 M	79.4 ms	78.2
AdaptViG-S (Ours) w/ KNN	8.6 M	71.2 ms	78.9
AdaptViG-S (Ours) w/ AGC	**8.6 M**	**42.3 ms**	**79.6**

B.3. Throughput Comparisons

To validate the practical efficiency of our model, we measure the inference throughput. We compare our AdaptViG models against other Vision GNNs (PViG [15], PViHGNN [16]) and State Space Models (Vim) [68] in Figure 5.

As shown in Figure 5, our AdaptViG models clearly establish a new state-of-the-art Pareto frontier, outperforming all competing architectures. Notably, our most efficient model, AdaptViG-S (79.6%), achieves a throughput of over 3000 images/sec, which is more than twice as fast as Vim-S [68] (80.3%) and approximately three times as fast as PViG-S [15] (82.1%). This demonstrates the superior efficiency of our AdaptViG architecture.

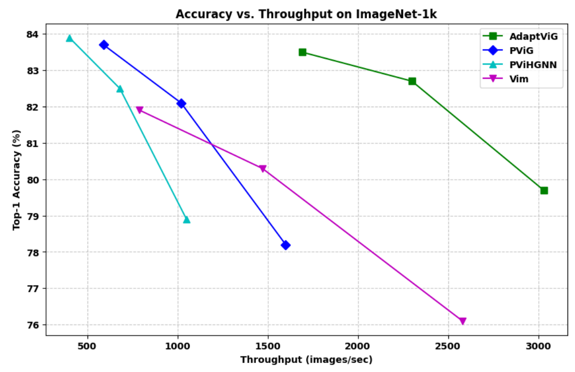


Figure 5. **Accuracy vs. Throughput comparison on ImageNet-1k.** We plot Top-1 Accuracy against inference throughput (images/sec). AdaptViG clearly establishes a new state-of-the-art Pareto frontier, demonstrating significantly higher throughput and accuracy than competing models.

B.4. Ablation on Gating Temperature (T)

To analyze the impact of our learnable temperature parameter T, we conduct an ablation study presented in Table 7. We compare our standard model, which learns T per layer, against variants with a fixed, non-learnable T set to different values. The results show that making T learnable yields the best performance (83.3%). While the model is reasonably robust to different fixed values, a very low temperature (e.g., $T = 0.5$) is overly restrictive and hurts performance the most, while a higher temperature (e.g., $T = 2.0$) can be too permissive. This validates our choice to make the temperature an adaptive, learnable parameter.

Table 7. **Ablation on the temperature parameter (T)** in AdaptViG-B. Our learnable setting is optimal. The Top-1 accuracy results are averaged over two experiments.

Configuration	Params (M)	Top-1 (%)
Fixed $T = 0.5$	26.8	83.1
Fixed $T = 1.0$	26.8	83.2
Fixed $T = 2.0$	26.8	83.2
Learnable T (Ours)	**26.8**	**83.3**

Learned Temperature Ablation. To analyze the behavior of our learnable temperature parameter, we inspect its final converged value for our AdaptViG-B model. The average learned value for each stage is reported in Table 8.

The results in Table 8 reveal a clear trend across the network's depth. In **Stage 1**, the temperature converges to a higher value (3.93). A high temperature makes the gate highly permissive, suggesting that in early stages where features are low-level (e.g., edges, colors), the model learns to aggregate a wide range of information without aggressive pruning.

Table 8. **Average learned temperature (T) values** per stage for the **AdaptViG-B** model after 300 epochs of training on ImageNet-1K.

Model	**Stage 1**	**Stage 2**	**Stage 3**
AdaptViG-B	3.93	0.68	1.11

In **Stage 2**, as features become more semantically meaningful, the temperature drops significantly to 0.68. This makes the gate more selective, allowing the model to form sparser, more content-aware connections. Finally, in **Stage 3**, the temperature stabilizes at 1.11, very near its initialization value of 1.0 detailed in Section C.2. This suggests that a temperature around 1.0 represents a stable, optimal trade-off for gating the highly abstract features in the deepest layers. This trend demonstrates that our gating mechanism adaptively learns the appropriate level of sparsity for features at different semantic levels, validating the robustness of our approach.

B.5. Ablation on Gating Function Design

To justify our choice of using the L1 distance and an exponential decay function for our gate, we compare it against other plausible designs in Table 9. Using the L2 distance, which is more sensitive to outliers, results in a 0.1% accuracy drop. Replacing the exponential decay with a standard sigmoid gate leads to a more significant 0.3% drop. These results confirm that our proposed combination of L1 distance for robust feature comparison and exponential decay for stable gating is the most effective design.

Table 9. **Ablation on the gating function design** in AdaptViG-B. Our L1 + Exponential Decay design is superior.

Distance Metric	**Gating Function**	**Top-1 (%)**
L2 Norm	Exponential Decay	83.2
L1 Norm	Sigmoid	83.0
L1 Norm (Ours)	**Exponential Decay (Ours)**	**83.3**

B.6. Ablation on Feature Fusion Strategy

Our AGC module fuses information from neighbors using a max-relative difference operation following prior work [3, 41]. In Table 10, we compare this against several common alternatives: simple addition of neighbor features, channel-wise concatenation, and sum aggregation used in Graph Isomorphism Networks (GIN) [60]. Relying on simple addition leads to a 0.5% drop in accuracy, likely due to feature saturation. Using only concatenation performs better but is still 0.4% worse than our method. Comparing to sum aggregation from GINs [60] our approach yields a +0.2% improvement. This validates that the max-relative difference operation is the most effective fusion strategy for our architecture.

Table 10. **Ablation on the feature fusion strategy** in AdaptViG-B, following the graph convolution step.

Fusion Strategy	**Top-1 (%)**
Simple Addition	82.8
Concatenation	82.9
GIN (Sum Aggregation) [60]	83.1
Max-Relative Difference (Ours)	**83.3**

C. Implementation Details

C.1. Network Configuration

The detailed network architectures for AdaptViG-S, M, and B are provided in Table 11. We report the configuration of the stem, the four stages, and the final classification head. In each stage, we list the number of local Inverted Residual Blocks (IRBs) and global processing blocks (either AGC or Attention), as well as the channel dimensions (C).

Table 11. **Architecture details of AdaptViG.** showing configuration of the stem, stages, and classification head. C represents the channel dimensions.

Stage	AdaptViG-S	AdaptViG-M	AdaptViG-B
Stem	Conv $\times 2$,	Conv $\times 2$,	Conv $\times 2$,
Stage 1	IRB Block $\times$ 3 AGC Block $\times$ 3 $C = 32$	IRB Block $\times$ 4 AGC Block $\times$ 4 $C = 48$	IRB Block $\times$ 5 AGC Block $\times$ 5 $C = 48$
Stage 2	IRB Block $\times$ 3 AGC Block $\times$ 3 $C = 64$	IRB Block $\times$ 4 AGC Block $\times$ 4 $C = 96$	IRB Block $\times$ 5 AGC Block $\times$ 5 $C = 96$
Stage 3	IRB Block $\times$ 9 AGC Block $\times$ 3 $C = 128$	IRB Block $\times$ 12 AGC Block $\times$ 4 $C = 192$	IRB Block $\times$ 15 AGC Block $\times$ 5 $C = 192$
Stage 4	IRB Block $\times$ 3 Attention Block $\times$ 3 $C = 256$	IRB Block $\times$ 4 Attention Block $\times$ 4 $C = 320$	IRB Block $\times$ 5 Attention Block $\times$ 5 $C = 384$
Head	Pooling & MLP		

C.2. Hyperparameter Settings

The detailed hyperparameter settings used for our ImageNet-1K training are provided in Table 12. The hyperparameter settings match those of Vision GNN [15], MobileViG [41], and EfficientFormer [31] for fair comparison.

Table 12. **Training hyperparameters for ImageNet-1K.**

Hyperparameter	Value
Epochs	300
Optimizer	AdamW [36]
Batch Size	1024
Start Learning Rate (LR)	$2e^{-3}$
LR Schedule	Cosine
Warmup Epochs	20
Weight Decay	0.05
Repeated Augment [19]	✓
RandAugment [10]	✓
Mixup Prob. [65]	0.8
Cutmix Prob. [64]	1.0
Random Erasing Prob. [66]	0.25
Exponential Moving Average	0.99996
Temperature (T) Initialization	1.0

D. Additional Experimental Results

D.1. CIFAR-100 Image Classification Results

We conduct image classification experiments on the CIFAR-100 [25] dataset, training from scratch for 200 epochs. We report the top-1 accuracy on the test set and implement all models using PyTorch [45] and the Timm library [58] with the AdamW [36] optimizer and a cosine annealing schedule.

Table 13. **Results of our AdaptViG-S and competing methods on the CIFAR-100** image classification task.

Model	Type	Params (M)	Top-1 (%)
ResNet-50 [17]	CNN	23.7	80.9
ConvNeXt-T [35]	CNN	28.0	82.5
MobileViG-Ti [41]	CNN-GNN	4.3	80.2
MobileViG-B [41]	CNN-GNN	25.4	83.8
Swin-T [34]	ViT	28.0	74.9
AdaptViG-S (Ours)	**CNN-GNN**	**7.5**	**84.0**

As shown in Table 13, our AdaptViG-S model achieves state-of-the-art performance among efficient models on CIFAR-100. With only 7.5M parameters, it obtains 84.0% Top-1 accuracy, outperforming MobileViG-B by 0.2% while using 71% fewer parameters, demonstrating its superior efficiency.

D.2. CIFAR-10 Image Classification Results

We conduct further image classification experiments on the CIFAR-10 [25] dataset, which consists of 10 object classes, training from scratch for 200 epochs.

Table 14. **Results of our AdaptViG-S and competing methods on the on the CIFAR-10** image classification task.

Model	Type	Params (M)	Top-1 Acc (%)
ConvNeXt-T [35]	CNN	28.0	97.1
MobileViG-Ti [41]	CNN-GNN	4.3	95.6
MobileViG-B [41]	CNN-GNN	25.3	96.7
Swin-T [34]	ViT	28.0	91.1
AdaptViG-S (Ours)	**CNN-GNN**	**7.5**	**97.0**

When tested on CIFAR-10, as shown in Table 14, AdaptViG-S continues to show superior performance. It achieves 97.0% Top-1 accuracy, which is 5.9% higher than Swin-T with 73% fewer parameters. Our model also achieves virtually the same state-of-the-art performance as ConvNeXt-T (97.0% vs. 97.1%) while using only a fraction of the parameters (7.5M vs. 28.0M), a 73% reduction.

D.3. Medical Image Classification Results

We evaluate AdaptViG on two distinct medical image classification tasks from the MedMNISTv2 benchmark [61].

The first, OrganSMNIST, consists of 11 organ classes from 13,932 training and 2,452 validation abdominal CT images. The second, DermaMNIST, consists of 7 skin lesion classes from 7,007 training and 1,003 validation Dermatoscope images. For both datasets, we train models from scratch for 200 epochs using the AdamW optimizer [36] and a cosine annealing schedule. All implementations use PyTorch [45] and the Timm library [58].

Table 15. **Results on the OrganSMNIST** medical image classification task.

Model	Type	Params (M)	Top-1 Acc (%)
ResNet-50 [17]	CNN	23.7	92.6
ConvNeXt-T [35]	CNN	28.0	91.6
MobileViG-Ti [41]	CNN-GNN	4.3	90.7
Swin-T [34]	ViT	28.0	91.7
AdaptViG-S (Ours)	**CNN-GNN**	**7.5**	**92.0**

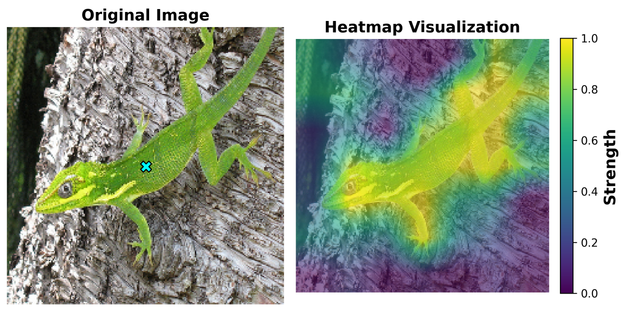
OrganSMNIST. On the OrganSMNIST dataset, as shown in Table 15, our AdaptViG-S model demonstrates superior performance and efficiency. It achieves a Top-1 accuracy of **92.0%**, which is higher than both Swin-T [34] (+0.3%) and ConvNeXt-T [35] (+0.4%) while using approximately 73% fewer parameters than both. This result highlights our model's ability to learn effective representations for volumetric medical data.

Table 16. **Results on the DermaMNIST** medical image classification task.

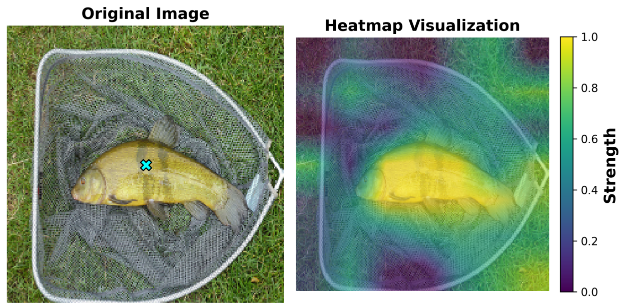
Model	Type	Params (M)	Top-1 Acc (%)
ResNet-50 [17]	CNN	23.7	76.1
ConvNeXt-T [35]	CNN	28.0	78.1
MobileViG-Ti [41]	CNN-GNN	4.3	75.4
Swin-T [34]	ViT	28.0	80.0
AdaptViG-S (Ours)	**CNN-GNN**	**7.5**	**76.1**

DermaMNIST. When tested on the DermaMNIST dataset, our AdaptViG-S continues to show a strong efficiency profile. As shown in Table 16, it achieves 76.0% accuracy, outperforming the lighter MobileViG-Ti [41] by 0.7%. While Swin-T [34] and ConvNeXt-T [35] achieve higher absolute accuracy, our model provides competitive performance with significantly fewer parameters, underscoring its utility in resource-constrained medical imaging applications.

E. Qualitative Visualizations



(a) Standard case with a clear object and background.



(b) Challenging case with similar color object and background.

Figure 6. **Visualization of learned spatial interaction strength.** From a reference point on the primary object (cyan 'X'), we visualize the learned weights the model assigns across the image. Warmer colors (yellow) indicate higher interaction strength. The model correctly focuses on the object in both a standard case (lizard) and a more challenging one with similar object and background colors (fish), demonstrating its ability to learn robust, object-aware features.

To provide insight into our model's learned behavior, we perform a qualitative analysis by visualizing the learned interaction strength within one of the model's global blocks. In Figure 6, for a selected reference point on the primary object (marked with a cyan 'X'), we compute and overlay a heatmap of the learned spatial weights across the image.

The visualizations demonstrate that our model learns to focus on semantically coherent regions, validating our design. In the standard case (a), the reference point on the lizard's body yields high interaction scores across the lizard's entire body, including its head and tail, while correctly suppressing the distinct tree background. In the more challenging case (b), where the fish's color is similar to the surrounding grass, the model still demonstrates a strong ability to focus on the coherent shape of the fish. This shows that our architecture learns robust feature representations that can identify objects from their backgrounds.