Appendix

The appendix is organized as follows:

- Sec. A1 provides the complete proofs for the claims stated in the main paper.
- Sec. A2 includes additional implementation, runtime details, and memory consumption.
- Sec. A3 fully describes the augmentation settings used for image classification and semantic segmentation.
- Sec. A4 provides additional semantic segmentation qualitative results.
- Sec. A5 reports additional image classification results covering:
  a. The robustness of our proposed models to out-of-distribution images (Sec. A5.1).
  b. The sensitivity of our ViT models to input shifts of different magnitudes (Sec. A5.2).
  c. The use of our proposed adaptive modules on pre-trained ViTs (Sec. A5.3).

A1. Complete proofs

A1.1. Proof of Lemma 1

Lemma 1. *L-periodic shift-equivariance of tokenization.*
Let input $\boldsymbol{x} \in \mathbb{R}^N$ have a token representation $\boldsymbol{X}^{(m)}\boldsymbol{E} \in \mathbb{R}^{\lfloor N/L \rfloor \times D}$. If $\hat{\boldsymbol{x}} = \mathcal{S}_N \boldsymbol{x}$ (a shifted input), then its token representation $\hat{\boldsymbol{X}}^{(m)}\boldsymbol{E}$ corresponds to:

$$\hat{\boldsymbol{X}}^{(m)}\boldsymbol{E} = \mathcal{S}^{\lfloor (m+1)/L \rfloor}_{\lfloor N/L \rfloor} \boldsymbol{X}^{((m+1) \bmod L)}\boldsymbol{E}. \tag{15}$$

This implies that $\boldsymbol{x}$ and $\hat{\boldsymbol{x}}$ are characterized by the same L token representations, up to a circular shift along the token index (row index of $\boldsymbol{X}^{((m+1) \bmod L)}\boldsymbol{E}$).

Proof. By definition:

$$\hat{\boldsymbol{X}}^{(m)} = \texttt{reshape}(\mathcal{S}^m_N \hat{\boldsymbol{x}}) = \texttt{reshape}(\mathcal{S}^{m+1}_N \boldsymbol{x}). \tag{A27}$$

Let the input patches be expressed as $\boldsymbol{X}^{(m)} = \begin{bmatrix} \boldsymbol{r}^{(m)}_0 & \dots & \boldsymbol{r}^{(m)}_{L-1} \end{bmatrix} \in \mathbb{R}^{\lfloor N/L \rfloor \times L}$, where $\boldsymbol{r}^{(m)}_k \in \mathbb{R}^{\lfloor N/L \rfloor}$ is comprised by the k^{th} element of every input patch:

$$\boldsymbol{r}^{(m)}_k[n] = (\mathcal{S}^m_N \boldsymbol{x})[Ln + k] = x[(Ln + k + m) \bmod N]. \tag{A28}$$

More precisely, given a circularly shifted input with offset m, $\boldsymbol{r}^{(m)}_k[n]$ represents the k^{th} element of the n^{th} patch. Based on this, $\hat{\boldsymbol{X}}^{(m)} \in \mathbb{R}^{\lfloor N/L \rfloor \times L}$ can be expressed as:

$$\hat{\boldsymbol{X}}^{(m)} = \texttt{reshape}(\mathcal{S}^{m+1}_N \boldsymbol{x}) = \begin{bmatrix} \boldsymbol{r}^{(m+1)}_0 & \dots & \boldsymbol{r}^{(m+1)}_{L-1} \end{bmatrix}, \tag{A29}$$

$$\text{with } \boldsymbol{r}^{(m+1)}_k[n] = \boldsymbol{x}[(Ln + k + m + 1) \bmod N]. \tag{A30}$$

Expressed in terms of its quotient and remainder for divisor L, $m + 1 = \lfloor \frac{m+1}{L} \rfloor L + (m + 1) \bmod L$. Then, $\hat{\boldsymbol{r}}$ corresponds to:

$$\hat{\boldsymbol{r}}^{(m)}_k[n] = \boldsymbol{x}\left[\left(L\left(n + \lfloor \frac{m+1}{L} \rfloor\right) + (m + 1) \bmod L + k\right) \bmod N\right]. \tag{A31}$$

It follows that $\hat{\boldsymbol{r}}^{(m)}_k = \mathcal{S}^{\lfloor \frac{m+1}{L} \rfloor}_{\lfloor \frac{N}{L} \rfloor} \boldsymbol{r}^{((m+1) \bmod L)}_k$, which implies:

$$\hat{\boldsymbol{X}}^{(m)}\boldsymbol{E} = \mathcal{S}^{\lfloor \frac{m+1}{L} \rfloor}_{\lfloor \frac{N}{L} \rfloor} \boldsymbol{X}^{((m+1) \bmod L)}\boldsymbol{E}. \tag{A32}$$

□

A1.2. Proof of Claim 1

Claim 1. *Shift-equivariance of adaptive tokenization.*
If F in Eq. (14) is shift-invariant, then `A-token` *is shift-equivariant, i.e., $\exists\, m_q \in \{0, \dots, L-1\}$ s.t.*

$$\texttt{A-token}\big(\mathcal{S}_N \boldsymbol{x}\big) = \mathcal{S}^{m_q}_{\lfloor N/L \rfloor} \texttt{A-token}(\boldsymbol{x}). \tag{16}$$

Proof. Let $\hat{\boldsymbol{x}} = \mathcal{S}_N \boldsymbol{x} \in \mathbb{R}^N$ be a circularly shifted version of input $\boldsymbol{x} \in \mathbb{R}^N$. From Lemma 1, their token representations satisfy:

$$\hat{\boldsymbol{X}}^{(m)} \boldsymbol{E} = \mathcal{S}^{\lfloor \frac{m+1}{L} \rfloor}_{\lfloor \frac{N}{L} \rfloor} \boldsymbol{X}^{((m+1) \bmod L)} \boldsymbol{E}. \tag{A33}$$

Based on the selection criterion of `A-token` and Eq. (A33):

$$\max_{m \in \{0,\dots,L-1\}} F(\hat{\boldsymbol{X}}^{(m)} \boldsymbol{E}) = \max_{m \in \{0,\dots,L-1\}} F\left(\mathcal{S}^{\lfloor \frac{m+1}{L} \rfloor}_{\lfloor \frac{N}{L} \rfloor} \boldsymbol{X}^{((m+1) \bmod L)} \boldsymbol{E}\right) \tag{A34}$$

$$= \max_{m \in \{0,\dots,L-1\}} F\left(\boldsymbol{X}^{((m+1) \bmod L)} \boldsymbol{E}\right), \tag{A35}$$

where the right-hand side in Eq. (A35) derives from the shift-invariance property of F. Notice that for any integer m in the range $\{0, \dots, L-1\}$, the circular shift $(m+1) \bmod L$ also lies within the same range. It follows that:

$$\max_{m \in \{0,\dots,L-1\}} F\left(\boldsymbol{X}^{((m+1) \bmod L)} \boldsymbol{E}\right) = \max_{m \in \{0,\dots,L-1\}} F\big(\boldsymbol{X}^{(m)} \boldsymbol{E}\big). \tag{A36}$$

Next, let $\hat{\boldsymbol{X}}^{(\hat{m})} \boldsymbol{E} = \texttt{A-token}(\hat{\boldsymbol{x}})$. Then, from Lemma 1:

$$\max_{m \in \{0,\dots,L-1\}} F\big(\hat{\boldsymbol{X}}^{(m)} \boldsymbol{E}\big) = F\big(\hat{\boldsymbol{X}}^{(\hat{m})} \boldsymbol{E}\big) = F\big(\boldsymbol{X}^{((\hat{m}+1) \bmod L)} \boldsymbol{E}\big). \tag{A37}$$

Let $\boldsymbol{X}^{(m^\star)} \boldsymbol{E} = \texttt{A-token}(\boldsymbol{x})$. Then, From Eq. (A36) and Eq. (A37):

$$\max_{m \in \{0,\dots,L-1\}} F\big(\boldsymbol{X}^{(m)} \boldsymbol{E}\big) = F\big(\boldsymbol{X}^{((\hat{m}+1) \bmod L)} \boldsymbol{E}\big), \tag{A38}$$

which implies that $m^\star = (\hat{m}+1) \bmod L$. Finally, from Lemma 1 and Eq. (A38):

$$\texttt{A-token}(\mathcal{S}_N \boldsymbol{x}) = \mathcal{S}^{\lfloor \frac{(\hat{m}+1)}{L} \rfloor}_{\lfloor \frac{N}{L} \rfloor} \boldsymbol{X}^{((\hat{m}+1) \bmod L)} \boldsymbol{E} = \mathcal{S}^{\lfloor \frac{(\hat{m}+1)}{L} \rfloor}_{\lfloor \frac{N}{L} \rfloor} \texttt{A-token}(\boldsymbol{x}). \tag{A39}$$

□

A1.3. Proof of Claim 2

Claim 2. *If G in Eq. (19) is shift invariant, then* `A-WSA` *is shift-equivariant.*

Proof. Let $\boldsymbol{T} \in \mathbb{R}^M$ and $\hat{\boldsymbol{T}} = \mathcal{S}_M \boldsymbol{T} \in \mathbb{R}^M$ denote two token representations related by a circular shift. From the definition of $\boldsymbol{v}^{(m)}_W$ in Eq. (17), let $\boldsymbol{v} \in \mathbb{R}^M$ denote the average ℓ_p-norm (energy) of each group of W neighboring tokens in $\boldsymbol{T}$. More precisely, the k-th component of the energy vector $\boldsymbol{v}$, denoted as $\boldsymbol{v}[k]$, is the energy of the window comprised by W neighboring tokens starting at index k:

$$\boldsymbol{v}[k] = \frac{1}{W} \sum_{l=0}^{W-1} \|\boldsymbol{T}_{(k+l) \bmod M}\|_p. \tag{A40}$$

Similarly, following Eq. (18), let $\boldsymbol{v}^{(m)}_W \in \mathbb{R}^{\lfloor \frac{M}{W} \rfloor}$ and $\hat{\boldsymbol{v}}^{(m)}_W \in \mathbb{R}^{\lfloor \frac{M}{W} \rfloor}$ denote the energy vectors of non-overlapping windows obtained after shifting $\boldsymbol{T}$ and $\hat{\boldsymbol{T}}$ by m indices, respectively. Then, maximizers $m^\star$ and $\hat{m}$ satisfy:

$$m^\star = \operatorname*{arg\,max}_{m \in \{0,\dots,W-1\}} G\big(\boldsymbol{v}^{(m)}_W\big), \;\; \hat{m} = \operatorname*{arg\,max}_{m \in \{0,\dots,W-1\}} G\big(\hat{\boldsymbol{v}}^{(m)}_W\big). \tag{A41}$$

Based on this, we prove that their adaptive window-based self-attention outputs are related by a circular shift that is a multiple of the window size W, *i.e.*, there exists $m_0 \in \mathbb{Z}$ such that:

$$\texttt{A-WSA}(\mathcal{S}_M \boldsymbol{T}) = \mathcal{S}_M^{m_0 W} \texttt{A-WSA}(\boldsymbol{T}). \tag{A42}$$

Given the shifted input token representation $\hat{\boldsymbol{T}} = \mathcal{S}_M \boldsymbol{T}$, the energy of each group of W neighboring tokens corresponds to:

$$\hat{\boldsymbol{v}}[k] = \frac{1}{W} \sum_{l=0}^{W-1} \|(\mathcal{S}_M \boldsymbol{T})_{(k+l) \bmod M}\|_p = \frac{1}{W} \sum_{l=0}^{W-1} \|\boldsymbol{T}_{(k+l+1) \bmod M}\|_p \tag{A43}$$

$$= \boldsymbol{v}[k+1], \tag{A44}$$

which implies $\hat{\boldsymbol{v}} = \mathcal{S}_M \boldsymbol{v}$. Then, $\hat{\boldsymbol{v}}_W^{(m)}$ can be expressed as:

$$\hat{\boldsymbol{v}}_W^{(m)}[k] = \hat{\boldsymbol{v}}[Wk + m] = \boldsymbol{v}[Wk + m + 1]. \tag{A45}$$

Expressing $m+1$ in terms of its quotient and remainder for divisor W:

$$\hat{\boldsymbol{v}}_M^{(m)}[k] = \boldsymbol{v}\left[W\big(k + \lfloor \frac{m+1}{W} \rfloor\big) + (m+1) \bmod W\right] \tag{A46}$$

$$= \mathcal{S}_{\lfloor \frac{M}{W} \rfloor}^{\lfloor \frac{m+1}{W} \rfloor} \boldsymbol{v}_W^{((m+1) \bmod W)}. \tag{A47}$$

Based on the shift-invariant property of G and the fact that $(m+1) \bmod W \in \{0, \dots, W-1\}$, A-WSA selection criterion corresponds to:

$$\max_{m \in \{0,\dots,W-1\}} G\big(\hat{\boldsymbol{v}}_W^{(m)}\big) = \max_{m \in \{0,\dots,W-1\}} G\big(\mathcal{S}_{\lfloor \frac{M}{W} \rfloor}^{\lfloor \frac{m+1}{W} \rfloor} \boldsymbol{v}_W^{((m+1) \bmod W)}\big) \tag{A48}$$

$$= \max_{m \in \{0,\dots,W-1\}} G\big(\boldsymbol{v}_W^{((m+1) \bmod W)}\big) \tag{A49}$$

$$= \max_{m \in \{0,\dots,W-1\}} G\big(\boldsymbol{v}_W^{(m)}\big). \tag{A50}$$

Then, from $\hat{m}$ in Eq. (A41):

$$\max_{m \in \{0,\dots,W-1\}} G\big(\hat{\boldsymbol{v}}_W^{(m)}\big) = G\big(\hat{\boldsymbol{v}}_W^{(\hat{m})}\big) = G\left(\boldsymbol{v}_W^{((\hat{m}+1) \bmod W)}\right), \tag{A51}$$

which implies $m^\star = (\hat{m}+1) \bmod W$. It follows that the adaptive self-attention of $\hat{\boldsymbol{T}}$ can be expressed as:

$$\texttt{A-WSA}(\hat{\boldsymbol{T}}) = \texttt{WSA}(\mathcal{S}_M^{\hat{m}} \hat{\boldsymbol{T}}) = \texttt{WSA}(\mathcal{S}_M^{\hat{m}+1} \boldsymbol{T}) \tag{A52}$$

$$= \texttt{WSA}\left(\mathcal{S}_M^{\lfloor (\hat{m}+1)/W \rfloor W + m^\star} \boldsymbol{T}\right) \tag{A53}$$

$$= \texttt{WSA}\left(\mathcal{S}_M^{\lfloor (\hat{m}+1)/W \rfloor W} \mathcal{S}_M^{m^\star} \boldsymbol{T}\right). \tag{A54}$$

Since $\mathcal{S}_M^{\lfloor (\hat{m}+1)/W \rfloor W}$ corresponds to a circular shift by a multiple of the window size W:

$$\texttt{A-WSA}(\hat{\boldsymbol{T}}) = \mathcal{S}_M^{\lfloor (\hat{m}+1)/W \rfloor W} \texttt{WSA}(\mathcal{S}_M^{m^\star} \boldsymbol{T}), \tag{A55}$$

where the right-hand side of Eq. (A55) stems from the fact that, for WSA with a window size W, circularly shifting the input tokens by a multiple of W results in an identical circular shift of the output tokens. Finally, from the definition of adaptive self-attention in Eq. (19):

$$\texttt{A-WSA}(\hat{\boldsymbol{T}}) = \mathcal{S}_M^{m_0 W} \texttt{A-WSA}(\boldsymbol{T}),\ m_0 = \lfloor (\hat{m}+1)/W \rfloor. \tag{A56}$$

□

Claim 2 shows that A-WSA induces an offset between token representations that is a multiple of the window size W. As a result, windows are comprised by the same tokens, despite circular shifts. This way, A-WSA guarantees that an input token representation and its circularly shifted version are split into the same token windows, leading to a circularly shifted self-attention output.

A1.4. Proof of Claim 3

Claim 3. *`PMerge` corresponds to a strided convolution with $\tilde{D}$ output channels, striding P and kernel size P.*

Proof. Let the input tokens be expressed as $\boldsymbol{T} = \begin{bmatrix}\boldsymbol{t}_0 & \dots & \boldsymbol{t}_{D-1}\end{bmatrix} \in \mathbb{R}^{M\times D}$, where $\boldsymbol{t}_j \in \mathbb{R}^M$ is comprised by the j^{th} element of every input token. This implies that, given the l^{th} token $\boldsymbol{T}_l \in \mathbb{R}^D$, $\boldsymbol{T}_l[j] = \boldsymbol{t}_j[l]$. Then, the patch merging output $\boldsymbol{Z} = \begin{bmatrix}\boldsymbol{z}_0 & \dots & \boldsymbol{z}_{\tilde{D}-1}\end{bmatrix}$ can be expressed as:

$$\boldsymbol{Z} = \texttt{PMerge}(\boldsymbol{T}) = \mathcal{D}^{(P)}(\boldsymbol{Y}) \in \mathbb{R}^{\frac{M}{P}\times\tilde{D}}, \tag{A57}$$

where $\mathcal{D}^{(P)} \in \mathbb{R}^{\frac{M}{P}\times M}$ is a downsampling operator of factor P, and $\boldsymbol{Y} = \begin{bmatrix}\boldsymbol{y}_0 & \dots & \boldsymbol{y}_{\tilde{D}-1}\end{bmatrix} \in \mathbb{R}^{M\times\tilde{D}}$ is the output of a convolution sum with kernel size P and $\tilde{D}$ output channels:

$$\boldsymbol{y}_k = \sum_{j=0}^{D-1} \boldsymbol{t}_j \circledast \boldsymbol{h}^{(k,j)} \in \mathbb{R}^M, \tag{A58}$$

where $\circledast$ denotes circular convolution and $\boldsymbol{h}^{(k,j)} = \begin{bmatrix}\boldsymbol{h}_0^{(k,j)} & \dots & \boldsymbol{h}_{P-1}^{(k,j)}\end{bmatrix}^\top \in \mathbb{R}^P$ denotes a convolutional kernel. Note that the convolution sum in Eq. (A58) involves $D\cdot\tilde{D}$ kernels, given that $k \in \{0,\dots,\tilde{D}-1\}$ and $j \in \{0,\dots,D-1\}$. Without loss of generality, assume the number of input tokens M is divisible by the patch length P. Then, due to the linearity of the downsampling operator, $\boldsymbol{z}_k \in \mathbb{R}^{\frac{M}{P}}$ corresponds to:

$$\boldsymbol{z}_k = \sum_{j=0}^{D-1} \mathcal{D}^{(P)}\left(\boldsymbol{t}_j \circledast \boldsymbol{h}^{(k,j)}\right). \tag{A59}$$

Let $\boldsymbol{H}^{(k,j)} \in \mathbb{R}^{M\times M}$ be a convolutional matrix representing kernel $\boldsymbol{h}^{(k,j)}$:

$$\boldsymbol{H}^{(k,j)} = \begin{bmatrix} \boldsymbol{h}_0^{(k,j)} & \cdots & \boldsymbol{h}_{P-1}^{(k,j)} & \\ & \boldsymbol{h}_0^{(k,j)} & \cdots & \boldsymbol{h}_{P-1}^{(k,j)} \\ & & \ddots & \\ \cdots & \boldsymbol{h}_{P-1}^{(k,j)} & \cdots & \boldsymbol{h}_0^{(k,j)} \end{bmatrix}. \tag{A60}$$

Then, note that each summation term in Eq. (A59) can be expressed as a matrix-vector multiplication:

$$\boldsymbol{z}_k = \sum_{j=0}^{D-1} \tilde{\boldsymbol{H}}^{(k,j)}\boldsymbol{t}_j, \tag{A61}$$

where $\tilde{\boldsymbol{H}}^{(k,j)} = \mathcal{D}^{(P)}\boldsymbol{H}^{(k,j)} \in \mathbb{R}^{\frac{M}{P}\times M}$ corresponds to the convolutional matrix $\boldsymbol{H}^{(k,j)}$ downsampled by a factor P along the row index:

$$\tilde{\boldsymbol{H}}^{(k,j)} = \begin{bmatrix} \boldsymbol{h}_0^{(k,j)} & \cdots & \boldsymbol{h}_{P-1}^{(k,j)} & & & & & \\ & & & \boldsymbol{h}_0^{(k,j)} & \cdots & \boldsymbol{h}_{P-1}^{(k,j)} & & \\ & & & & & \ddots & & \\ & & & & & & \boldsymbol{h}_0^{(k,j)} & \cdots & \boldsymbol{h}_{P-1}^{(k,j)} \end{bmatrix}. \tag{A62}$$

Based on this, the convolution sum in Eq. (A59) can be expressed in matrix-vector form as:

$$\boldsymbol{z}_k = \begin{bmatrix}\tilde{\boldsymbol{H}}^{(k,0)} & \cdots & \tilde{\boldsymbol{H}}^{(k,D-1)}\end{bmatrix} \begin{bmatrix}\boldsymbol{t}_0 \\ \vdots \\ \boldsymbol{t}_{D-1}\end{bmatrix}. \tag{A63}$$

Then, from the patch representation of $\boldsymbol{T}$ in Eq. (10), $\boldsymbol{z}_k$ can be alternatively expressed as:

$$\boldsymbol{z}_k = \tilde{\boldsymbol{T}}\tilde{\boldsymbol{e}}_k, \tag{A64}$$

where $\tilde{\boldsymbol{e}}_k$ is the k^{th} column of $\tilde{\boldsymbol{E}} = \begin{bmatrix}\tilde{\boldsymbol{e}}_0 & \cdots & \tilde{\boldsymbol{e}}_{\tilde{D}-1}\end{bmatrix} \in \mathbb{R}^{PD\times\tilde{D}}$. Let $\texttt{vec}(\bar{\boldsymbol{T}}_P^{(k)}) \in \mathbb{R}^{PD}$ operate in a column-wise manner:

$$\begin{aligned}
\texttt{vec}(\bar{\boldsymbol{T}}_P^{(k)}) &= \texttt{vec}\left(\begin{bmatrix}\boldsymbol{T}_{Pk} & \dots & \boldsymbol{T}_{P(k+1)-1}\end{bmatrix}^\top\right) && \text{(A65)}\\
&= \begin{bmatrix}\boldsymbol{T}_{Pk}[0] & \cdots & \boldsymbol{T}_{P(k+1)-1}[0] & \boldsymbol{T}_{Pk}[1] & \cdots & \boldsymbol{T}_{P(k+1)-1}[1] & \cdots & \boldsymbol{T}_{P(k+1)-1}[D-1]\end{bmatrix}^\top && \text{(A66)}\\
&= \begin{bmatrix}\boldsymbol{t}_0[Pk] & \cdots & \boldsymbol{t}_0[P(k+1)-1] & \boldsymbol{t}_1[Pk] & \cdots & \boldsymbol{t}_1[P(k+1)-1] & \cdots & \boldsymbol{t}_{D-1}[P(k+1)-1]\end{bmatrix}^\top. && \text{(A67)}
\end{aligned}$$

Finally, from Eq. (A63), $\tilde{\boldsymbol{e}}_k$ is equivalent to

$$\tilde{\boldsymbol{e}}_k = \begin{bmatrix}\boldsymbol{h}^{(k,0)} ; \dots ; \boldsymbol{h}^{(k,D-1)}\end{bmatrix} \in \mathbb{R}^{DP}, \tag{A68}$$

which shows that $\texttt{PMerge}(\boldsymbol{T}) = \tilde{\boldsymbol{T}}\tilde{\boldsymbol{E}} \in \mathbb{R}^{\frac{M}{P}\times\tilde{D}}$ can be expressed as a convolution sum with kernels comprised by columns of $\tilde{\boldsymbol{E}}$. □

Claim 3 shows that the original patch merging function `PMerge` is equivalent to projecting all M overlapping patches of length P, which can be expressed as a circular convolution, followed by keeping only $\frac{M}{P}$ of the resulting tokens. Note that the selection of such tokens is done via a downsampling operation of factor P, which is not the only way of selecting them. In fact, there are P different token representations that can be selected, as explained in Sec. 4.

Based on this observation, the proposed `A-PMerge` leverages the polyphase decomposition to select token representations in a data-adaptive manner, resulting in a circularly shift-equivariant patch merging scheme. Notably, while `A-PMerge` utilizes APS [3] and LPF [55] to inherently achieve shift equivariance (as explained in Section 4), the use of antialiasing filters is not strictly necessary for the core equivariance. The polyphase sampling itself guarantees perfect circular shift consistency. However, following the design of APS, we incorporate antialiasing filters in practice, as they have been shown to improve downstream task performance.

A2. Additional implementation details

We have attached our code in the supplementary materials. We now provide an illustration of how to use our implementation and verify that the proposed modules are indeed circularly shift-equivariant.

A2.1. `A-pmerge` usage

We show a toy example of how to use the adaptive patch merging layer (`A-pmerge`) by building a simple image classifier. The model is comprised by an `A-pmerge` layer, which includes a linear embedding, followed by a global average pooling layer and finally a linear classification head. We empirically show that this simple model is circularly shift-invariant.

```
# A-pmerge based classifier
class ApmergeClassifier(nn.Module):
  def __init__(
    self,stride,input_resolution,
    dim,num_classes=4,conv_padding_mode='circular'
  ):
    super().__init__()
    self.stride=stride
    # Pooling layer for A-pmerge
    pool_layer = partial(
      PolyphaseInvariantDown2D,
      component_selection=max_p_norm,
      antialias_layer=None,
    )
    # A-pmerge
```

```
    # No adaptive window selection
    # for illustration purposes
    self.apmerge = AdaptivePatchMerging(
      input_resolution=input_resolution,
      dim=dim,
      pool_layer=pool_layer,
      conv_padding_mode=conv_padding_mode,
      stride=stride,
      window_selection=None,
      window_size=None,
    )
    # Global pooling and head
    self.avgpool=nn.AdaptiveAvgPool2d((1,1))
    self.fc=nn.Linear(dim*stride,num_classes)

  def forward(self,x):
    # Reshape
    B,C,H,W = x.shape
    x = x.permute(0,2,3,1).reshape(B,H*W,C)
    # Adaptive patch merge
    x = self.apmerge(x)
    # Reshape back
    x = x.reshape(
      B,H//self.stride,W//self.stride,C*self.stride,
    ).permute(0,3,1,2)
    # Global average pooling
    x = torch.flatten(self.avgpool(x),1)
    # Classification head
    x = self.fc(x)
    return x

# Input tokens
B,C,H,W = 1,3,8,8
stride = 2
x = torch.randn(B,C,H,W).cuda().double()
# Shifted input
shift = torch.randint(-3,3,(2,))
x_shift = torch.roll(input=x,dims=(2,3),shifts=(shift[0],shift[1]))
# A-pmerge classifier
model = ApmergeClassifier(
  stride=stride,
  input_resolution=(H,W),
  dim=C).cuda().double().eval()
# Predict
y = model(x)
y_shift = model(x_shift)
err = torch.norm(y-y_shift)
assert(torch.allclose(y,y_shift))
# Check circularly shift invariance
print('y: {}'.format(y))
print('y_shift: {}'.format(y_shift))
print("error: {}".format(err))
```

Out:

```
y: tensor([[-0.1878,  0.2348,  0.0982, -0.2191]],
    device='cuda:0', dtype=torch.float64,
    grad_fn=<AddmmBackward0>)
y_shift: tensor([[-0.1878,  0.2348,  0.0982, -0.2191]],
    device='cuda:0', dtype=torch.float64,
    grad_fn=<AddmmBackward0>)
error: 0.0
```

A2.2. Simple encoder-decoder usage

To illustrate the use of our adaptive layers for semantic segmentation purposes, we demonstrate the use of the `A-PMerge`'s optimal indices and the `A-WSA`'s optimal offset for unpooling purposes. We build a simple encoder-decoder based on an `A-PMerge` layer equipped with `A-WSA` (implemented as `poly_win` in the example code) to encode input tokens, followed by an unpooling module to place features back into their original positions at high resolution.

This example depicts the strategy used by our proposed semantic segmentation models, where backbone indices are passed to the segmentation head to upscale feature maps in a consistent fashion, leading to a circularly shift-equivariant model.

```python
# Simple encoder-decoder
class EncDec(nn.Module):
  def __init__(self,dims,stride,win_sel,win_sz,pad):
    super().__init__()
    self.pad = pad
    self.stride = stride
    self.win_sz = win_sz

    # A-pmerge equipped with A-WSA
    pool_layer = get_pool_method(name='max_2_norm',antialias_mode='skip')
    self.apmerge = AdaptivePatchMerging(
      dim=dims,
      pool_layer=pool_layer,
      conv_padding_mode=pad,
      stride=stride,
      window_size=win_sz,
      window_selection=win_sel,
    )
    # Unpool
    unpool_layer = get_unpool_method(unpool=True,pool_method='max_2_norm',
        antialias_mode='skip')
    self.unpool = set_unpool(unpool_layer=unpool_layer,stride=stride,
        p_ch=dims)

  def forward(self,x,hw_shape):
    B,C,H,W = x.shape
    # Reshape
    x = x.permute(0,2,3,1).reshape(B,H*W,C)
    # A-pmerge + A-WSA
    x_w,hw_shape,idx = self.apmerge(x=x,hw_shape=hw_shape,ret_indices=True)
    # Keep merging and windowing indices
    idx = idx[:3]
    # Reshape back
    x_w = x_w.reshape(
```

```
      B,H//self.stride,W//self.stride,C*self.stride).permute(0,3,1,2)
    #  Unroll and Unpool
    y = unroll_unpool_multistage(
      x=x_w,
      scale_factor=[self.stride],
      unpool_layer=self.unpool,
      idx=[idx],
      unpool_winsel_roll=self.pad,
    )
    return y

# Input
B,C,H,W = 1,3,28,28
shift_max = 6
x = torch.randn(B,C,H,W).cuda().double()
# Offsets
s01 = torch.randint(low=-shift_max,high=shift_max,size=(1,2)).tolist()[0]
s02 = torch.randint(low=-shift_max,high=shift_max,size=(1,2)).tolist()[0]
s03 = [s02[0]-s01[0],s02[1]-s01[1]]
# Shifted inputs
x01 = torch.roll(x,shifts=s01,dims=(-1,-2))
x02 = torch.roll(x,shifts=s02,dims=(-1,-2))
# Build encoder-decoder
# Use A-WSA (poly_win)
model = EncDec(dims=3,stride=2,pad='circular',win_sz=7,win_sel=poly_win,
    ).cuda().double().eval()
# Predictions
y01 = model(x01,hw_shape=(H,W))
y02 = model(x02,hw_shape=(H,W))
# Shift to compare
z = torch.roll(y01,shifts=s03,dims=(-1,-2))
err = torch.norm(z-y02)
assert torch.allclose(z,y02)
print("torch.norm(z-y02): {}".format(err))
```

Out:

```
torch.norm(z-y02): 0.0
```

A2.3. Memory Consumption

We compare the computational requirements of our adaptive Vision Transformer models and their default versions in terms of training and inference GPU memory. We report memory consumption on a single NVIDIA Quadro RTX 5000 GPU, where both training and inference are performed using batch size 64 and the default image size per model.

Following previous work [12, 24, 25], we compare the *allocated* GPU memory required by each model via NVIDIA's `torch.cuda.max_memory_allocated()` command. This measures the maximum GPU memory occupied by tensors since the beginning of the executed program. Additionally, we report the maximum *reserved* GPU memory, as assigned by the CUDA memory allocator, using the `torch.cuda.max_memory_reserved()` command. This measures the maximum memory occupied by tensors plus the extra memory reserved to speed up allocations. The required memory is reported in Megabytes (1 MiB $= 2^{20}$ bytes). We also report the relative change, which corresponds to the memory increase with respect to the default models.

Tab. A1 shows the memory required for training and inference for each of our adaptive models and their default versions.

Model	Training		Inference	
	Max. Allocated Memory (MiB)	Max. Reserved Memory (MiB)	Max. Allocated Memory (MiB)	Max. Reserved Memory (MiB)
Swin-T	4,935	5,380	1,012	1,352
A-Swin-T **(Ours)**	5,178 (+4.92%)	5,678 (+5.54%)	1,012 (0%)	1,382 (+2.22%)
SwinV2-T	7,712	8,016	1,275	1,544
A-SwinV2-T **(Ours)**	8,113 (+5.2%)	8,710 (+8.66%)	1,275 (0%)	1,534 (−0.65%)
CvT-13	5,794	6,044	1,587	1,850
A-CvT-13 **(Ours)**	6,257 (+7.99%)	6,528 (+8.01%)	1,587 (0%)	1,780 (−3.78%)
MViTv2-T	6,335	7,352	1,944	3,356
A-MViTv2-T **(Ours)**	7,364 (+16.24%)	8,456 (+15.02%)	2,164 (+11.32%)	3,378 (+0.67%)

Table A1. **Memory Consumption:** Maximum allocated and reserved memory required by our adaptive ViTs and their default versions. Training and inference memory consumption is calculated on a single NVIDIA Quadro RTX 5000 GPU (batch size 64, default image size per model), and reported in Megabytes (MiB). We also report the relative change with respect to the default models (%), which is shown in parentheses.

	Train Set	Test Set
Size Preprocessing	(i) Resize (Swin, MViTv2, CvT: 256×256. SwinV2: 292×292) (ii) Center Crop (Swin, MViTv2, CvT: 224×224, SwinV2: 256×256)	
Augmentation	(i) Random Horizontal Flipping (ii) Normalization	Normalization

Table A2. **Image Classification preprocessing (circular shifts).** Data preprocessing and augmentation used to train and test all default and adaptive ViTs under circular shifts.

In terms of training, our adaptive framework marginally increases the allocated memory requirements up to 16% for MViTv2, while the rest of the models increase the memory requirements by less than 8%. This trend is also seen in the reserved memory, where it increases up to 15%. Additionally, in terms of inference, only our adaptive MViTv2 model marginally increases the allocated memory by 11%, while the rest of our adaptive models do not increase the required allocated memory. In terms of reserved memory, our adaptive models increase the requirement at most by 2%, while in some cases the reserved memory decreases by 3%.

Overall, the training memory consumption marginally increases with respect to the default models, while the inference memory consumption remains almost unaffected.

A3. Additional experimental details

A3.1. Image classification

Data pre-processing (ImageNet circular shifts, CIFAR10/100). Similar to previous work on circularly shift-equivariant CNNs [3, 34], to highlight the fact that perfect shift-equivariance is imposed by design and not induced during training, no shift or resizing augmentation is applied. Tab. A2 includes the data pre-processing used in our ImageNet experiments under circular shifts, and in all our CIFAR10/100 experiments. As explained in Sec. 5.2, input images are transformed to match the default image size used by each model. This process is denoted as *Size pre-processing* in the table.

Data pre-processing (ImageNet standard shifts). In contrast to the circular shift scenario, each model uses its default data augmentation configuration for the ImageNet standard shift scenario. Tables A3, A4 and A5 include the augmentation details of each model, A-Swin, A-SwinV2, A-MViTv2, and A-CVT respectively. Similar to the circular shift case, all models use a data pre-processing step to change the input image size.

Note that all four models use RandAugment [6] as data augmentation pipeline, which includes: `AutoContrast`, `Equalize`, `Invert`, `Rotate`, `Posterize`, `Solarize`, `ColorIncreasing`, `ContrastIncreasing`, `BrightnessIncreasing`, `SharpnessIncreasing`, `Shear-x`, `Shear-y`, `Translate-x`, and `Translate-y`.

Computational settings. CIFAR10/100 classification experiments using all four models were trained on two NVIDIA Quadro RTX 5000 GPUs with a batch size of 48 images for 100 epochs. On the other hand, ImageNet classification experiments were trained on eight NVIDIA A100 GPUs, where all models used their default effective batch size and numerical precision for 300 epochs.

	Train Set	Test Set
Size Preprocessing	(i) Resize (Swin: 256×256, SwinV2: 292×292) (ii) Center Crop (Swin: 224×224, SwinV2: 256×256)	
Augmentation	(i) Random Horizontal Flipping (ii) RandAugment (magnitude: 9 increase severity: True, augmentations per image: 2, standard deviation: 0.5) (iii) Normalization (iv) RandomErasing	Normalization

Table A3. **Swin / SwinV2 Image Classification preprocessing (standard shifts).** Data preprocessing and augmentation used to train and test the default (Swin, SwinV2) and adaptive (A-Swin, A-SwinV2) models.

	Train Set	Test Set
Size Preprocessing	(i) Resize (256×256) (ii) Center Crop (224×224)	
Augmentation	(i) Random Horizontal Flipping (ii) RandAugment (magnitude: 10 increase severity: True, augmentations per image: 6, standard deviation: 0.5) (iii) Normalization (iv) RandomErasing	Normalization

Table A4. **MViTv2 Image Classification preprocessing (standard shifts).** Data preprocessing and augmentation used to train and test the default (MViTv2) and adaptive (A-MViTv2) models.

	Train Set	Test Set
Size Preprocessing	(i) Resize (256×256) (ii) Center Crop (224×224)	
Augmentation	(i) Random Horizontal Flipping (ii) RandAugment (magnitude: 9 increase severity: True, augmentations per image: 2, standard deviation: 0.5) (iii) Normalization (iv) RandomErasing	Normalization

Table A5. **CvT Image Classification preprocessing (standard shifts).** Data preprocessing and augmentation used to train and test the default (CvT) and adaptive (A-CvT) models.

A3.2. Semantic Segmentation

Data pre-processing (Circular shifts). For Swin+UperNet, both the baseline and adaptive (A-Swin) models were trained by resizing input images to size $1,792 \times 448$ followed by a random cropping of size 448×448. Next, the default data augmentation used in the Swin + UperNet model is used. During testing, input images are resized to 1792×448. Then, each dimension size is rounded up to the next multiple of 224. Tab. A6 describes the pre-processing and data augmentation pipelines.

For Swin-V2+UperNet, both the baseline and the adaptive (A-SwinV2) models were trained by resizing input images to size $2,048 \times 512$ followed by a random cropping of size 512×512. Following this, the default data augmentation used in the

Swin + UperNet model is adopted. During testing, input images are resized to $2,048 \times 512$. Then, each dimension size is rounded up to the next multiple of 256. Tab. A7 describes the pre-processing and data augmentation pipelines.

	Train Set	Test Set
Size Preprocessing	(i) Resize: $1,792 \times 448$ (ii) Random Crop: 448×448	(i) Resize: $1,792 \times 448$ (ii) Resize to next multiple of 224
Augmentation	(i) Random Horizontal Flipping (ii) Photometric Distortion (iii) Normalization	Normalization

Table A6. **Swin + UperNet Semantic Segmentation preprocessing (circular shifts).** Data preprocessing and augmentation used to train and test the default (Swin+UperNet) and adaptive (A-Swin+UperNet) models.

	Train Set	Test Set
Size Preprocessing	(i) Resize: $2,048 \times 512$ (ii) Random Crop: 512×512	(i) Resize: $2,042 \times 512$ (ii) Resize to next multiple of 256
Augmentation	(i) Random Horizontal Flipping (ii) Photometric Distortion (iii) Normalization	Normalization

Table A7. **SwinV2 + UperNet Semantic Segmentation preprocessing (circular shifts).** Data preprocessing and augmentation used to train and test the default (SwinV2+UperNet) and adaptive (A-SwinV2+UperNet) models.

Data pre-processing (Standard Shifts). For the Swin+UperNet baseline and adaptive models, the default pre-processing pipeline from the official MMseg implementation [5] was used. The same pipeline was used to evaluate the SwinV2+UperNet baseline and adaptive models. The pre-processing pipeline is detailed in Tab. A8.

Computational settings. ADE20K semantic segmentation experiments using Swin and SwinV2 models, both adaptive and baseline architectures, were trained on four NVIDIA A100 GPUs with an effective batch size of 16 images for $160,000$ iterations. These settings are consistent with the official MMSeg configuration for the Swin+UperNet model.

A4. Additional semantic segmentation results

Fig. A1 and A2 show examples of semantic segmentation predicted masks for our A-Swin+UperNet and A-SwinV2+Upernet models, respectively. Illustrations include masks obtained with their corresponding baselines, showing the improved robustness of our models against input shifts.

A5. Additional image classification experiments

A5.1. Robustness to out of distribution images

Similar to work in network robustness [15, 35, 38, 53], we evaluate the image classification performance of our adaptive models on out-of-distribution inputs. More precisely, we measure the robustness of our proposed A-Swin-T model to images with randomly erased patches and vertically flipped images.

Experiment Setup. Following the evaluation protocol of Chaman and Dokmanic [3], we compare the performance of three versions of Swin-T: (i) its default model, (ii) our proposed adaptive model (A-Swin-T), and (iii) the default Swin-T architecture trained on circularly shifted images, denoted as Swin-T DA (data augmented).

All three models are trained on CIFAR-10. Swin-T and A-Swin-T use the default data pre-processing used for CIFAR-10 (refer to Sec. A3), while Swin-T DA uses the default pre-processing plus circular shifts with offsets uniformly selected between -224 and 224 pixels. Note that CIFAR-10 samples are resized from 32×32 to 224×224 pixels during pre-processing. So, the circular offsets are effectively selected between -32 and 32 pixels.

During testing, the patch size is randomly selected between 0 and max, where max $\in \{28, 42, 56, 70\}$ pixels. Since CIFAR-10 samples are rescaled to 224 x 224 pixels, the maximum patch sizes are effectively $\{4, 6, 8, 10\}$ pixels.

Results. Tab. A9 shows the top-1 classification accuracy and circular shift consistency of the three Swin-T models of interest on images with randomly erased patches. Across patch sizes, while the default model and Swin-T DA decrease their shift

	Train Set	Test Set
Size Preprocessing	(i) Resize: $2,048 \times 512$ (ii) Random Crop: 512×512	Resize: $2,048 \times 512$
Augmentation	(i) Random Horizontal Flipping (ii) Photometric Distortion (iii) Normalization	Normalization

Table A8. **Semantic Segmentation preprocessing (standard shifts).** Data preprocessing and augmentation used to train and test the default and adaptive versions of both SwinV2+UperNet and A-SwinV2+UperNet models.

Model	max = 0		max = 28		max = 42		max = 56		max = 70	
	Acc.	C-Cons.	Top-1 Acc.	C-Cons.	Top-1 Acc.	C-Cons.	Top-1 Acc.	C-Cons.	Top-1 Acc.	C-Cons.
Swin-T (Default)	90.21	82.69	90.16	81.75	89.71	81.42	89.02	80.4	87.83	79.58
Swin-T DA	92.32	94.3	91.81	94.19	91.6	94.16	90.42	93.01	89.23	92.73
A-Swin-T **(Ours)**	**93.53**	**100**	**93.27**	**100**	**92.71**	**100**	**91.57**	**100**	**90.3**	**100**

Table A9. **Performance on images with randomly erased patches.** Top-1 classification accuracy (%) and circular shift consistency (%) on CIFAR-10 test images with randomly erased square patches. Max corresponds to the largest possible patch size, as sampled from a uniform distribution $\mathcal{U}\{0, \max\}$.

Model	Unflipped		Flipped	
	Top-1 Acc.(%)	C-Cons.(%)	Top-1 Acc.(%)	C-Cons.(%)
Swin-T (Default)	90.21	82.69	50.41	82.24
Swin-T DA	92.32	94.3	51.74	94.67
A-Swin-T **(Ours)**	**93.53**	**100**	**52.07**	**100**

Table A10. **Performance on flipped images.** Top-1 classification accuracy and circular shift consistency on vertically flipped CIFAR-10 test images.

consistency by at least 1.5%, our A-Swin-T preserves its perfect circular shift consistency. Our adaptive model also gets the best classification accuracy in all cases, improving by more than 1%. This suggests that, despite not being explicitly trained on this transformation, our A-Swin model is more robust than the default and DA models, obtaining better accuracy and consistency across scenarios.

On the other hand, Tab. A10 shows the circular shift consistency and classification accuracy of the three Swin models evaluated under vertically flipped images. Even in such a challenging case and without any fine-tuning, our A-Swin-T model retains its perfect circular shift consistency, improving over the default and data-augmented models by at least 5%. Our adaptive model also outperforms the default and data-augmented models in terms of classification accuracy, both on flipped and unflipped images, by a significant margin.

A5.2. Sensitivity to input shifts

To measure the shift consistency improvement of our proposed adaptive models over the default ones, we conduct a fine-grained evaluation of shift consistency by analyzing different offset magnitudes.

Experiment setup. We test the shift sensitivity of our four proposed adaptive ViTs on CIFAR-10. Circular shift consistency is evaluated on images shifted by an offset randomly selected from four different ranges: $[0, 56]$, $[0, 112]$, $[0, 168]$, and $[0, 224]$ pixels. Note that CIFAR-10 images are resized from 32×32 to 224×224 pixels. So, the effective intervals correspond to $[0, 8]$, $[0, 16]$, $[0, 24]$, and $[0, 32]$ pixels, respectively. By gradually increasing the offset magnitude, we can understand the advantages of our adaptive models over their default versions in a more general manner.

Results. Tab. A11 shows the circular shift consistency of all four models under different shift magnitudes. While the default versions monotonically decrease their shift consistency with respect to the offset magnitude, our method shows a perfect shift consistency across scenarios.

Despite the strong consistency obtained by default ViTs via data augmentation, our adaptive models outperform them by more than 9% without any fine-tuning. This shows the benefits of our adaptive models, regardless of the shift magnitude.

Model	Offset $\in \{0, \dots, 8\}$	Offset $\in \{0, \dots, 16\}$	Offset $\in \{0, \dots, 24\}$	Offset $\in \{0, \dots, 32\}$
Swin-T	$92.00 \pm .23$	$89.65 \pm .9$	$88.93 \pm .09$	$88.13 \pm .14$
A-Swin-T **(Ours)**	**100**	**100**	**100**	**100**
SwinV2-T	$92.13 \pm .04$	$90.43 \pm .17$	$89.67 \pm .08$	$88.75 \pm .15$
A-SwinV2-T **(Ours)**	**100**	**100**	**100**	**100**
CvT-13	$88.99 \pm .1$	87.42 ± 0.16	$86.96 \pm .1$	$86.84 \pm .06$
A-CvT-13 **(Ours)**	**100**	**100**	**100**	**100**
MViTv2-T	$91.49 \pm .04$	$90.57. \pm .11$	$90.46 \pm .08$	$90.22 \pm .1$
A-MViTv2-T **(Ours)**	**100**	**100**	**100**	**100**

Table A11. **Consistency under different shift magnitudes.** Circular shift consistency (%) of our adaptive ViT models under small, medium, large, and very large shifts. Models trained and evaluated on CIFAR-10 under a circular shift assumption.

Model	Top-1 Acc.(%)	C-Cons.(%)
CvT-13 (Default)	90.12	76.54
CvT-13 + Adapt (No Fine-tuning)	57.4	100
CvT-13 + Adapt (10 epoch Fine-tuning)	91.72	100
CvT-13 + Adapt (20 epoch Fine-tuning)	92.35	100
A-CvT-13 **(Ours)**	**93.87**	**100**

Table A12. **Incorporating shift-equivariant modules on pre-trained ViTs.** Our shift-equivariant ViT framework allows plugging-in shift equivariant modules on pre-trained models (e.g. CvT-13), improving on classification accuracy after a few fine-tuning iterations while preserving its perfect shift consistency. Results shown on CvT-13 trained on CIFAR-10 under a circular shift assumption.

A5.3. Replacing adaptive modules in pre-trained ViTs

We ran additional experiments plugging in our proposed adaptive modules on pre-trained default ViTs. Notice that, while replacing default ViT modules with adaptive ones guarantees a circularly shift-equivariant model (100% shift consistency), it is expected for the classification accuracy to decrease since the pre-trained weights have not been trained to adaptively select different token or window representations. Nevertheless, we are interested in exploring this scenario as a refined initialization strategy, in order to evaluate its effect in terms of classification accuracy and shift consistency.

Experiment setup. Given a default CvT-13 model trained on CIFAR-10, we replace its components with our proposed framework. We denote this model as CvT-13 + Adapt. Next, we fine-tune the model for 10 and 20 epochs and evaluate its top-1 classification accuracy as well as circular shift consistency.

Results. Tab. A12 shows the top-1 classification accuracy and circular shift consistency of CvT-13 + Adapt, as well as the default CvT-13 and our proposed A-CvT-13 model. Without any fine-tuning, CvT-13 + Adapt attains perfect circular shift consistency just by plugging in our proposed adaptive modules. However, its top-1 classification accuracy decreases by more than 30% with respect to the default model. By fine-tuning the model for 10 and 20 epochs, its classification accuracy boosts up to 91.72% and 92.35%, respectively. This implies that CvT-13 + Adapt is capable of outperforming the default model both in consistency and accuracy by plugging in our adaptive components plus a few fine-tuning iterations.

On the other hand, while promising results are obtained via fine-tuning, our adaptive model A-CvT-13, fully trained for 90 epochs using random initialization and the default training settings, still attains the best classification accuracy and circular shift consistency results.

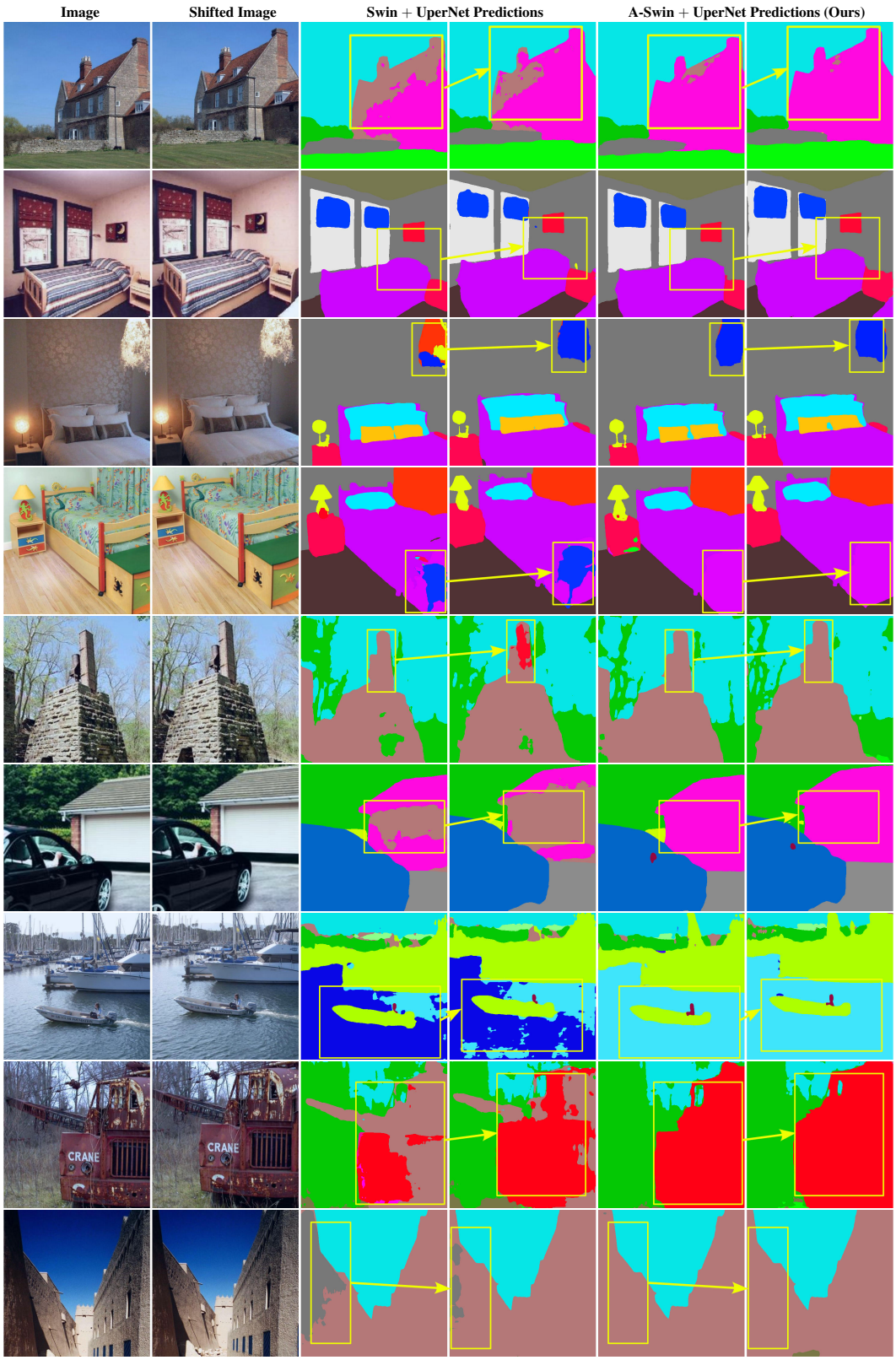

Figure A1. **Swin + UperNet Semantic Segmentation under standard shifts:** Semantic segmentation results on the ADE20K dataset (standard shifts) via Swin backbones: Our A-Swin + UperNet model is more robust to input shifts than the original Swin + UperNet model, generating consistent predictions while improving accuracy. Examples of prediction changes due to input shifts are boxed in yellow.

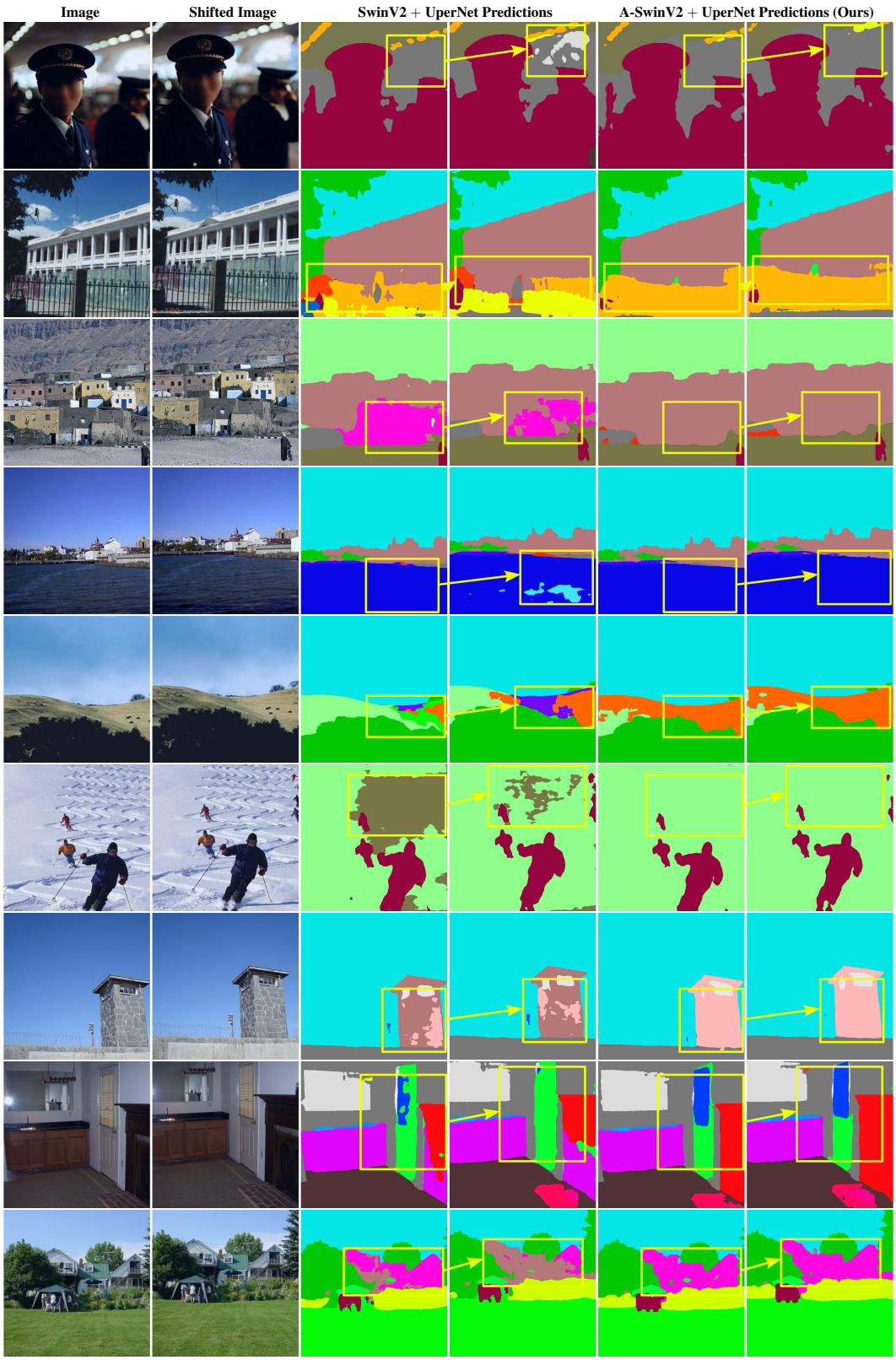

Figure A2. **SwinV2 + UperNet Semantic Segmentation under standard shifts:** Additional semantic segmentation results on the ADE20K dataset (standard shifts) via SwinV2 backbones: Our adaptive model improves both segmentation accuracy and shift consistency with respect to the original SwinV2 + UperNet model. Examples of prediction changes due to input shifts are boxed in yellow.