Swin Transformer Masking — Complete Explanation

---

1. What is the goal of Swin Transformer?

In ViT, every token attends to every other token across the entire image.
This gives global understanding but costs $O(n^2)$ — very expensive for high-res images.

Swin Transformer computes attention ONLY within local windows → O(n) per window.
But then how does the model understand the full image?
→ Through SHIFTED WINDOWS across layers.

---

2. What happens in Regular Window Partition?

Feature map (8×8) → divided into non-overlapping windows (four 4×4 windows).
Self-attention computed independently inside each window.

- Tokens in Window A → only see Window A tokens
- Tokens in Window B → only see Window B tokens
- ZERO communication between windows

Problem: Hard boundaries.
A token at the edge of Window A cannot attend to its immediate neighbor in Window B.

---

3. Why do we shift windows?

In the next layer, shift the window grid by (M/2, M/2) — half the window size.
Window boundaries now fall at different positions.

Tokens that were separated by a boundary before → now inside the SAME window.
They can finally attend to each other.

Shifting = the mechanism that creates CROSS-WINDOW CONNECTIONS.
Without shifting, each window = isolated island forever.

---

4. What problem does shifting create?

After shifting, border windows become IRREGULAR in size.
Instead of 4 clean 4×4 windows → 9 sub-regions of different sizes:

- Some 4×4
- Some 2×4

- Some 4×2
- Some 2×2

Computing attention on 9 differently-sized regions = terrible for GPU efficiency.
GPUs need UNIFORM shape for batching and parallelization.

---

5. What is Cyclic Shift and why is it used?

To fix irregular windows → use CYCLIC SHIFT (torch.roll).

Take border fragments sticking out → wrap them to the opposite side.

- Top rows → move to bottom
- Left columns → move to right

Like rolling a sheet of paper into a cylinder.

Result: Back to exactly 4 windows of size 4×4 → uniform, GPU-friendly, efficient.

---

6. What NEW problem does Cyclic Shift introduce?

CRITICAL PART:

After cyclic shift, some 4×4 windows contain tokens from
COMPLETELY DIFFERENT spatial locations in the original image.

Example:

- Tokens from TOP-LEFT corner of image
- Tokens from BOTTOM-RIGHT corner of image
- Both end up in the SAME window

These tokens were NEVER spatial neighbors.
They have NO meaningful relationship.
They got grouped together PURELY as an artifact of the roll operation.

If we let them attend to each other → model learns FALSE RELATIONSHIPS.
"This pixel in top-left is related to that pixel in bottom-right"
→ This would CORRUPT the learned representations.

---

7. What does the Mask do?

Mask prevents false cross-region attention connections.

Before applying softmax, add a mask value to attention logits (raw QK^T/√d scores).

For every token pair (i, j) in a window, check:

- Did token i and token j come from the SAME original shifted window region?
- Or were they brought together ARTIFICIALLY by cyclic shift?

---

8. Why mask = 0 for same-region tokens?

Two tokens from SAME original region = genuine spatial neighbors.
They SHOULD attend to each other — this is the whole point of shifted window attention.

Adding 0 to their logit → changes ABSOLUTELY NOTHING.
Score remains whatever QK^T/√d computed.
Attention proceeds normally, as if no mask exists.

---

9. Why mask = −∞ for cross-region tokens?

Two tokens from DIFFERENT original regions = FAKE neighbors (created by cyclic shift).
We want their attention weight to be EXACTLY ZERO.

How −∞ achieves this:

softmax(x_i) = e^(x_i) / Σ e^(x_j)

When you add −∞ to a logit:
→ logit becomes −∞
→ e^(−∞) = 0 (exactly)
→ numerator = 0
→ softmax output = 0
→ attention weight = ZERO

That token pair contributes NOTHING to the output.

Meanwhile, valid token pairs (same-region):
→ softmax weights still SUM TO 1
→ proper probability distribution maintained
→ mathematically clean, no approximation, no leakage

---

10. Numerical Example of −∞ in Softmax

Pre-softmax logits = [2.0, 1.5, −∞, 0.8]

e^(2.0) = 7.39
e^(1.5) = 4.48

e^(−∞) = 0.00 ← EXACTLY ZERO
e^(0.8) = 2.23

Sum = 7.39 + 4.48 + 0.00 + 2.23 = 14.10

softmax = [0.524, 0.318, 0.000, 0.158]

→ Masked position gets EXACTLY zero weight
→ Remaining positions redistribute and still sum to 1

---

11. Why not multiply by 0 AFTER softmax?

If you multiply weights by 0 AFTER softmax:
→ remaining weights no longer sum to 1
→ probability distribution is BROKEN

By adding −∞ BEFORE softmax:
→ invalid positions eliminated DURING normalization
→ valid positions automatically redistribute to sum to 1

This is why ADDITIVE MASKING before softmax is the standard technique.
Same technique used for padding masks and causal masks in NLP transformers.

---

12. Why not just let ALL tokens attend to each other? Why block any?

Core confusion:
"If we want full image understanding, shouldn't every token know about every other token?"

Answer: Full-image understanding does NOT happen in a single layer.
It builds up GRADUALLY:

- Layer 1 (regular windows) → each token sees 4×4 local area
- Layer 2 (shifted windows) → receptive field roughly DOUBLES
- Layer 3 → doubles again
- Layer 4 → doubles again
- ... after several layers → receptive field covers ENTIRE IMAGE

Similar to CNNs:

- Early layers → small patches
- Deeper layers → whole image

The mask is NOT blocking useful long-range attention.
It is blocking MEANINGLESS ARTIFICIAL connections that cyclic shift accidentally created.

Those blocked tokens:

- Were NEVER spatial neighbors
- Are from OPPOSITE CORNERS of the image
- Will eventually connect through natural layer-by-layer growth
- NOT through this artificial shortcut

---

13. Complete Pipeline (Step by Step)

Step 1: Start with feature map

Step 2: Apply cyclic shift
→ torch.roll by −M/2 in both height and width

Step 3: Partition into regular 4×4 windows
→ Some windows now contain mixed-region tokens

Step 4: Compute attention with mask
→ QK^T/√d + b_ij + mask_ij
→ mask = 0 for same-region pairs
→ mask = −∞ for cross-region pairs

Step 5: Apply softmax
→ masked positions become exactly 0

Step 6: Multiply attention weights × Values
→ masked tokens contribute nothing

Step 7: Apply REVERSE cyclic shift
→ torch.roll by +M/2
→ put all tokens back to original spatial positions

Result: Mathematically equivalent to computing attention on irregular shifted windows,
but done efficiently using uniform window sizes + masking.

---

14. Summary Table

Situation	What is happening	Mask value
Same original shifted window	Genuine spatial neighbors, should attend	0
Different regions, cyclic shifted	Fake neighbors, artificially grouped by roll	−∞

---

15. Design Philosophy Summary

Regular window → gives EFFICIENCY
Shift → gives CROSS-WINDOW CONNECTIONS
Cyclic shift → restores UNIFORM WINDOW SIZES for GPU
Mask → prevents FALSE ATTENTION between accidentally grouped tokens
Alternating regular + shifted across layers → gradually builds GLOBAL RECEPTIVE FIELD

Same goal as ViT's global attention, but at a FRACTION of the cost.