MambaEye: A Size-Agnostic Visual Encoder with Causal Sequential Processing

Supplementary Material

1. Model Implementation

We provide the simplified PyTorch-like pseudo-code of the MambaEye model to clarify the architectural details. The model consists of an initial projection layer, a Mamba2 backbone, and a classification head.

```
class MambaEye(nn.Module):
    def __init__(self, num_classes, input_dim, dim, depth,
                 d_state, d_conv, expand):
        super().__init__()
        # Initial projection
        self.init_fc = nn.Sequential(
            nn.Linear(input_dim, dim),
            nn.SiLU()
        )

        # Mamba2 backbone with pre-norm architecture
        self.mamba2_net = Mamba2backbone(
            d_model=dim,
            n_layer=depth,
            ssm_cfg={
                "layer": "Mamba2",
                "d_state": d_state, # 128
                "d_conv": d_conv,   # 4
                "expand": expand,   # 2
            },
            rms_norm=True,
            fused_add_norm=True,
            residual_in_fp32=True
        )

        # Classification head
        self.classification_head = nn.Sequential(
            nn.Linear(dim, num_classes),
        )

    def forward(self, img_sequence, move_embedding):
        # Concatenate inputs: (B, L, D_img) + (B, L, D_move)
        x = torch.cat((img_sequence, move_embedding), dim=-1)

        # Initial projection
        x = self.init_fc(x)

        # Forward through Mamba2 backbone
        x, hidden_states_list = self.mamba2_net(x)

        # Classification head
        classification_sequence = self.classification_head(x)
```

```
        return classification_sequence
```

2. Data Processing Pipeline

Our data processing pipeline mimics saccadic eye movements by sampling patches from random locations. We provide the core logic for patch extraction and move embedding below.

2.1. Sequential Patch Sampling

```
def get_item(self, idx):
    # 1. Create a zero-padded canvas
    canvas = torch.ones(3, canvas_size, canvas_size) * padding_value

    # 2. Place image on canvas (centered)
    canvas[:, row_offset:row_offset+H, col_offset:col_offset+W] = img

    # 3. Randomly sample coordinates for the sequence
    # Sample top-left coordinates (r, c) for each step
    row_coords = randint(row_offset, row_end, sequence_length)
    col_coords = randint(col_offset, col_end, sequence_length)

    # 4. Extract patches
    patches = []
    for i in range(sequence_length):
        r, c = row_coords[i], col_coords[i]
        patch = canvas[:, r:r+patch_size, c:c+patch_size]
        patches.append(patch.flatten())

    # 5. Compute relative moves
    # delta_r[t] = r[t] - r[t-1] (with r[-1] = 0)
    delta_r = cat([row_coords[0:1], row_coords[1:] - row_coords[:-1]])
    delta_c = cat([col_coords[0:1], col_coords[1:] - col_coords[:-1]])

    # 6. Encode moves
    move_embeddings = sinusoidal_enc_2d(
        stack([delta_c, delta_r], dim=1), move_dim // 2
    )

    return stack(patches), move_embeddings
```

3. Training Logic

The training step involves computing the diffusion-inspired loss, which compares the model's predicted logits against a scheduled target distribution.

```
def training_step(self, batch):
    img_sequence, move_embedding, info_ratio, labels = batch

    # Forward pass
    logits_seq = self.model(img_sequence, move_embedding)

    # Calculate Loss
    # p_target: One-hot ground truth
    # p_prior: Uniform distribution
    # p_scheduled = (1 - r_t) * p_prior + r_t * p_target
```

```
    loss = 0
    for t in range(T):
        r_t = info_ratio[t]
        target_dist = (1 - r_t) * uniform_prior + r_t * one_hot_label
        loss += cross_entropy(logits_seq[:, t], target_dist)

    return loss / T
```

4. Additional Experimental Results

4.1. Loss Ablation Analysis

To further validate the effectiveness of our diffusion-inspired loss, we provide a detailed step-wise comparison against the standard Cross-Entropy (CE) loss baseline. Figure 1 illustrates the validation accuracy curves for both models across four different resolutions ($224^2, 512^2, 1024^2, 1536^2$) as a function of the number of steps processed.

4.2. Comprehensive Performance Analysis

We present the full performance curves for all trained MambaEye variants across different resolutions in Figure 2.

4.3. Scanning Path Analysis

We provide a detailed analysis of the scanning path ablation presented in the main paper. The key finding is that scan order performance is governed by how quickly the path exposes *global* image context to the causal model. We evaluated nine scanning patterns using MambaEye-B (FT) across resolutions from 224^2 to 2048^2.

Full Resolution Results. Table 1 presents the complete results. The patterns fall into two clear groups: *global-coverage* patterns (Golden, Random fixed grid) that maintain accuracy across resolutions, and *local-traversal* patterns (Column, Snake, Spiral, Hilbert, Diagonal, Raster) that collapse at high resolutions.

Local vs. Global Context. Deterministic paths that traverse mostly adjacent patches collapse at high resolutions because they lack early global context. In contrast, the **Golden** scan jumps forward by $\approx 61.8\%$ of the total sequence length at every step, guaranteeing that sample points are maximally spread across the entire image from the start. This provides global context without randomness and achieves performance comparable to random scanning.

Per-Pattern Step-Wise Analysis. Figure 3 shows the step-wise accuracy curves for each scanning pattern. Several notable behaviors emerge:

- **Golden and Random (fixed grid)** exhibit smooth, monotonically increasing curves resembling our default random sampling (cf. Figure 2), confirming that early global coverage is the key ingredient.
- **Column Raster and Horizontal Raster** show pronounced periodic oscillations caused by row/column boundary transitions, where large spatial jumps disrupt the causal state.
- **Snake variants** (Horizontal and Column) suppress the high-frequency oscillations by ensuring spatially continuous transitions at row/column boundaries, but still degrade because they only visit nearby patches in succession.
- **Diagonal** achieves the highest peak among local patterns at lower resolutions (54.7% at 384^2) because its traversal covers more spatial extent per step than row/column-based scans, but this advantage disappears at high resolutions.
- **Hilbert** curve, despite being a space-filling fractal designed for locality preservation, performs poorly because it still restricts the model to a local neighborhood for the first portion of the sequence.
- **Spiral** shows an interesting pattern: accuracy increases as the spiral expands outward from the center, providing gradually increasing spatial coverage, but collapses once the spiral wraps and re-enters previously visited regions.

Cross-Pattern Comparison. Figure 4 directly compares all patterns at four key resolutions. At 512^2, Golden and Random (fixed grid) reach $\sim 73\%$ while local patterns plateau between 31–51%. The gap widens dramatically at 1536^2, where global-coverage patterns maintain 66–68% accuracy while all local patterns fall below 13%.

4.4. Knowledge Distillation: Full Resolution Breakdown

Table 2 presents the complete knowledge distillation results across all evaluated resolutions. The step-wise accuracy progression is shown in Figure 2(g).

The KD model achieves the highest accuracy from 224^2 through 1024^2, with gains of $+0.4$ to $+1.9$ over the fine-tuned model. Only at 1408^2 and 1536^2 does the fine-tuned model pull slightly ahead ($+0.2$ points), as it was explicitly trained on longer sequences (T=2048). The KD model,

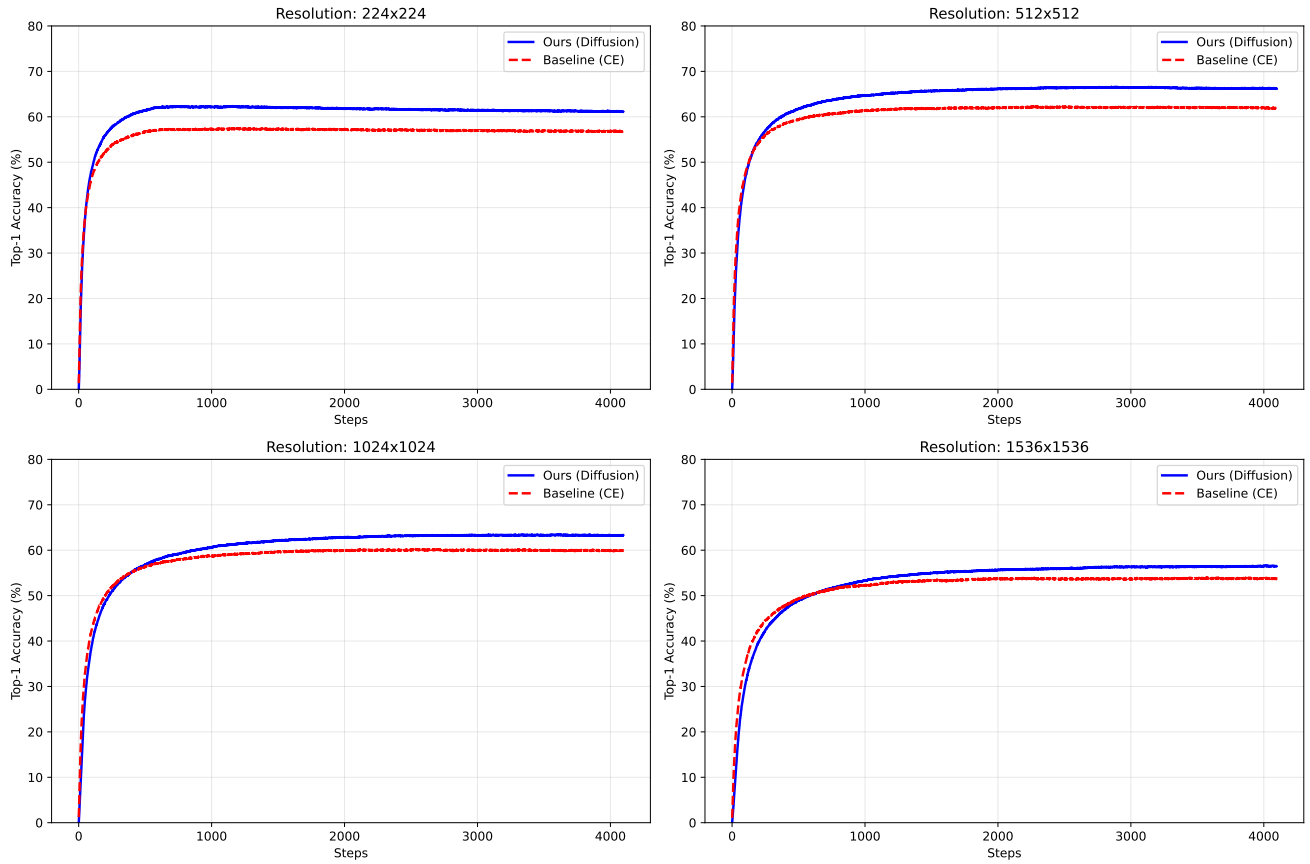


Figure 1. **Detailed Loss Ablation Comparison.** Step-wise top-1 accuracy comparison between MambaEye trained with our Diffusion-inspired loss (Blue) and the standard Cross-Entropy loss (Red dashed). Our method consistently achieves higher accuracy and faster convergence across all resolutions, particularly at higher resolutions where the dense supervision signal is most critical.

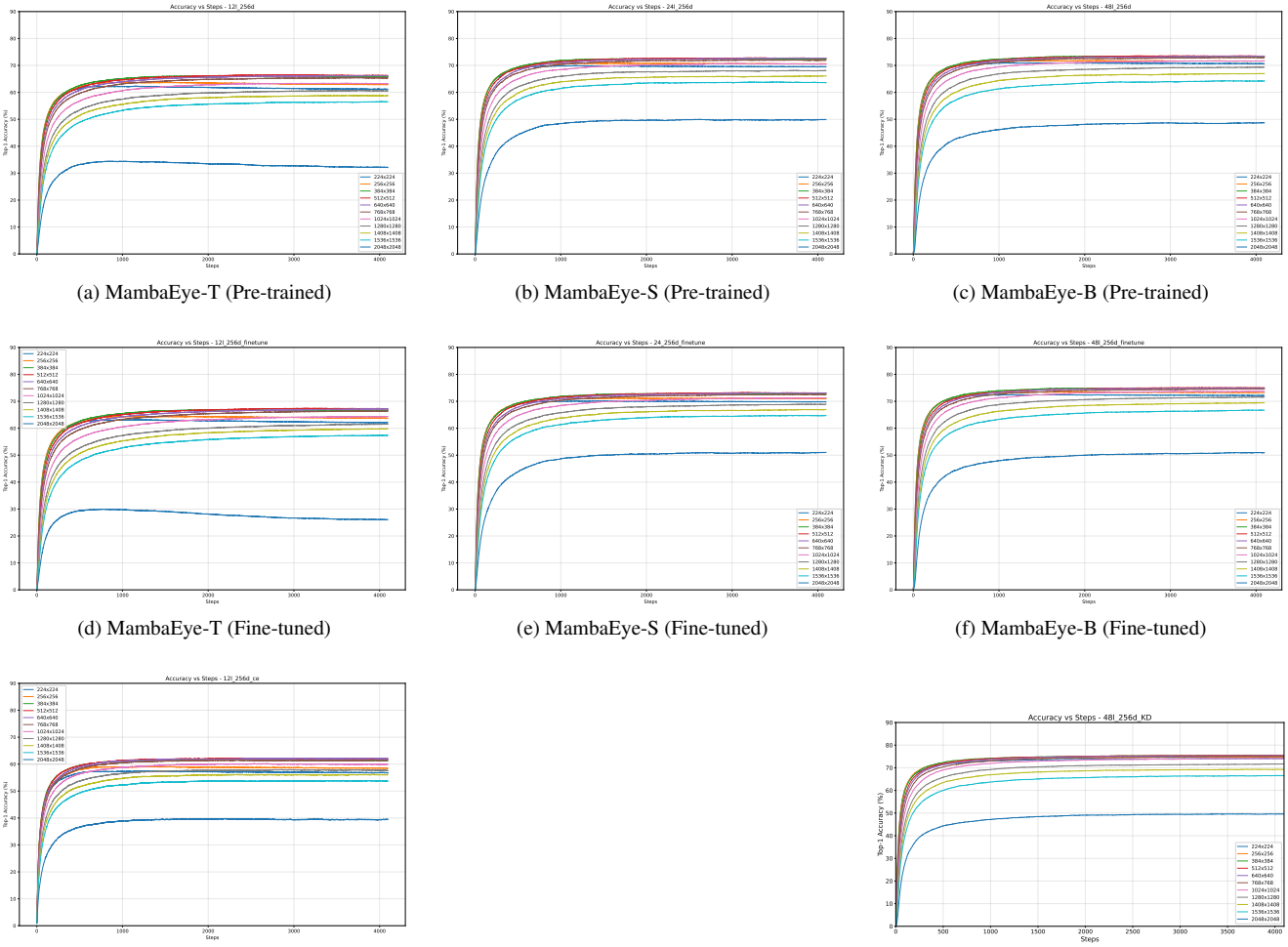
Table 1. Full scanning path results on ImageNet-1K using MambaEye-B (FT), T=4096. Top-1 accuracy (%).

Scanning Path	224^2	256^2	384^2	512^2	640^2	768^2	1024^2	1280^2	1536^2
Golden	59.4	64.8	71.5	73.2	73.7	73.7	73.1	70.8	**67.7**
Random (fixed grid)	**60.3**	**64.8**	**72.0**	**73.6**	**74.2**	**74.2**	**73.8**	**71.5**	66.5
Diagonal	51.2	53.5	54.7	51.4	47.0	39.8	26.0	14.2	9.4
Spiral	51.6	53.7	50.6	45.8	40.9	36.5	27.2	20.2	7.6
H. Raster	49.6	51.6	49.9	44.9	39.8	32.6	18.7	12.4	8.0
Hilbert	45.0	45.7	44.2	40.7	36.3	31.7	24.1	16.7	12.9
Column Raster	45.3	46.8	45.3	38.9	31.7	23.5	12.5	9.2	6.0
H. Snake	43.2	43.1	37.5	31.5	27.4	22.8	13.8	9.6	6.4
C. Snake	42.4	43.1	40.1	35.6	31.4	25.2	14.0	9.3	6.0

trained only at T=1024, generalizes well to 4× longer evaluation sequences. The teacher's knowledge transfers across resolutions: DeiT-III processes images only at 224^2, yet the student benefits at all resolutions, enabled by the information-ratio-weighted KD loss.

Table 2. Full KD results on ImageNet-1K (T=4096). Teacher: DeiT-III-Base (86M, 83.8%). Best per resolution in **bold**.

Model	224^2	256^2	384^2	512^2	640^2	768^2	1024^2	1280^2	1408^2	1536^2
Base	70.6	71.7	73.3	73.5	73.4	72.9	71.7	69.4	67.0	64.3
Base (FT)	72.2	73.2	74.8	75.0	75.0	74.6	73.7	71.5	**69.5**	**66.7**
Base (KD)	**74.1**	**74.8**	**75.5**	**75.5**	**75.4**	**75.0**	**74.1**	**71.7**	69.3	66.5

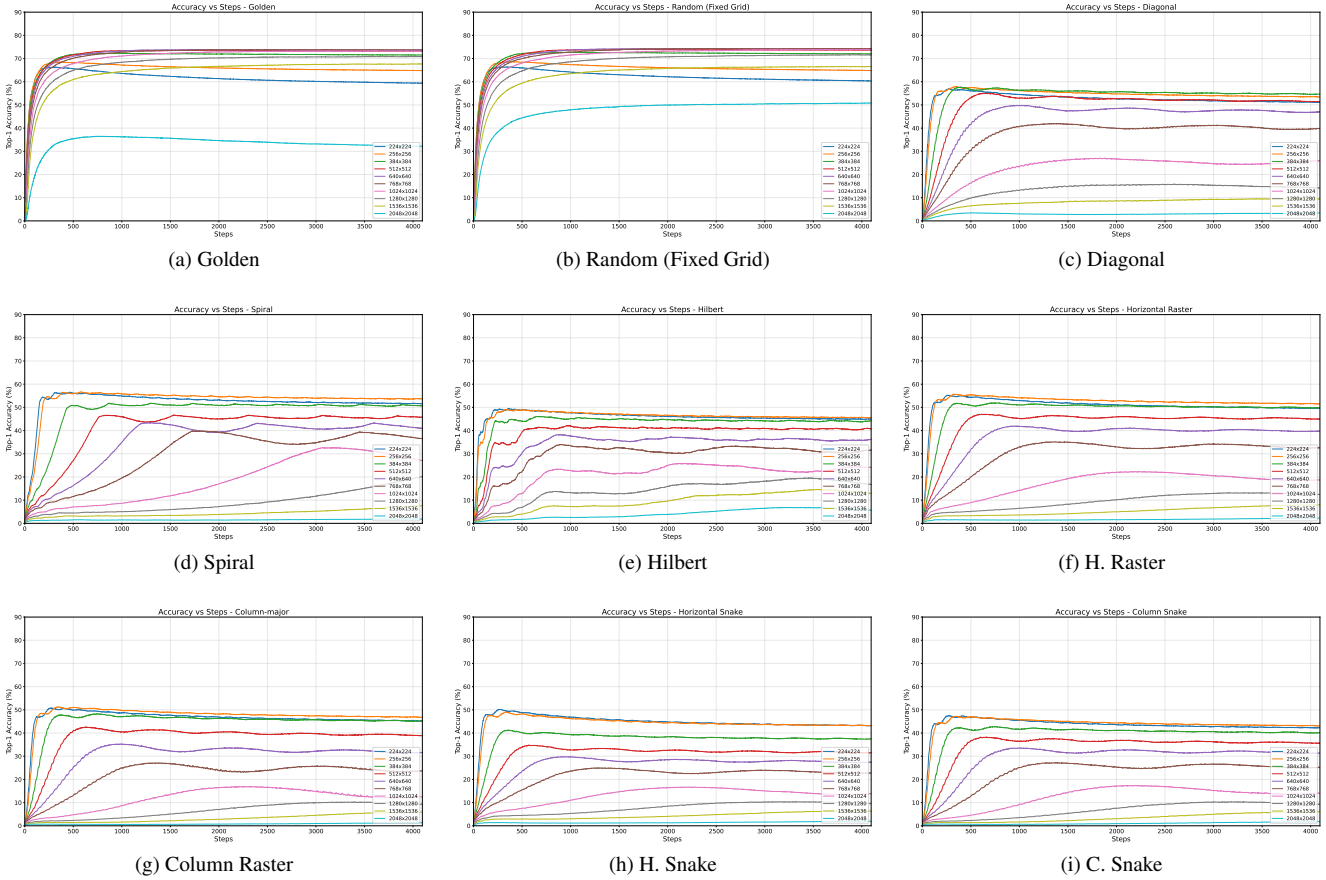


(a) MambaEye-T (Pre-trained) (b) MambaEye-S (Pre-trained) (c) MambaEye-B (Pre-trained)

(d) MambaEye-T (Fine-tuned) (e) MambaEye-S (Fine-tuned) (f) MambaEye-B (Fine-tuned)

(g) MambaEye-T (CE Loss Baseline) (h) MambaEye-B (KD)

Figure 2. **Comprehensive Performance Curves.** Accuracy vs. Steps for all MambaEye variants across multiple resolutions. Rows 1–2: pre-trained and fine-tuned models. Row 3: knowledge distillation and CE loss ablation.

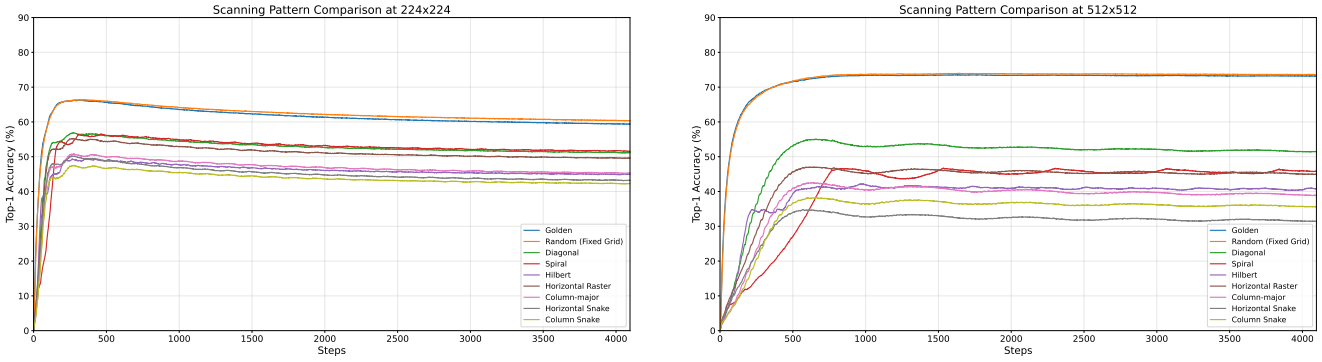


(a) Golden (b) Random (Fixed Grid) (c) Diagonal

(d) Spiral (e) Hilbert (f) H. Raster

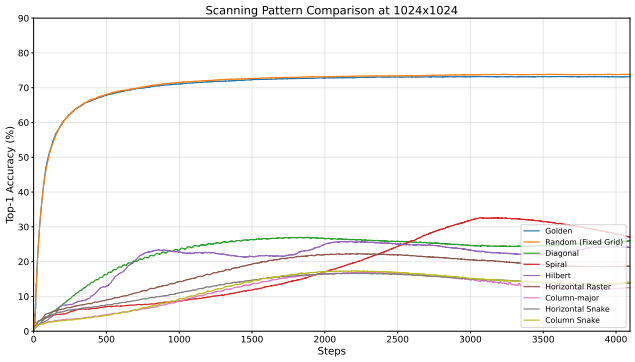
(g) Column Raster (h) H. Snake (i) C. Snake

Figure 3. **Step-wise accuracy for all scanning patterns.** Each subplot shows accuracy vs. sequence length (T=4096) for MambaEye-B (FT) across all resolutions. Global-coverage patterns (a–b) produce smooth, saturating curves. Local-traversal patterns (c–i) exhibit oscillations and degradation at high resolutions.

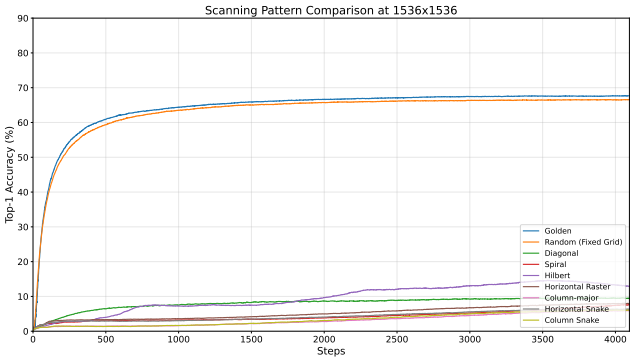


(a) 224^2

(b) 512^2



(c) 1024^2



(d) 1536^2

Figure 4. **Cross-pattern comparison at key resolutions.** All nine scanning patterns compared at 224^2, 512^2, 1024^2, and 1536^2. Global-coverage patterns (Golden, Random fixed grid) consistently outperform local-traversal patterns, with the gap widening at higher resolutions.