Group 1: What is SAM? (General Concepts)

1. **Question:** The SAM model requires a specific input resolution for its Image Encoder to function correctly with its pre-trained positional embeddings. What is this default resolution?

Answer: The model is designed to accept images resized to a standard resolution of 1024x1024 pixels. This is critical because the Vision Transformer (ViT) divides the image into patches, and the learned positional embeddings correspond to this specific grid size. If an image is not this size, it must be resized or padded before entering the encoder.

Source URL:
https://github.com/facebookresearch/segment-
anything/blob/main/segment_anything/modeling/image_encoder.py

Code Snippet:
```python
class ImageEncoderViT(nn.Module):
    def __init__(
        self,
        img_size: int = 1024,  # <--- Default resolution
        patch_size: int = 16,
```

2. **Question:** When a user clicks a single point on an object (eg. a shirt), it might be ambiguous (did they mean the shirt, the person, or just the pocket?). How does SAM handle this ambiguity by default?

Answer: To resolve ambiguity, SAM predicts **3 separate masks** for a single prompt. These typically correspond to three levels of granularity: the whole object, a part of the object, and a sub-part. This ensures that at least one of the masks is likely to be what the user intended.

Source URL:
https://www.google.com/search?q=https://github.com/facebookresearch/segment-
anything/blob/main/segment_anything/modeling/mask_decoder.py&authuser=1

Code Snippet:
```python
class MaskDecoder(nn.Module):
    def __init__(
        self,
        transformer_dim: int,
        transformer: nn.Module,
        num_multimask_outputs: int = 3, # <--- 3 outputs for ambiguity
```

3. **Question:** Neural networks use "activation functions" to introduce non-linearity. What specific activation function does the SAM architecture primarily use in its MLP (Multi-Layer Perceptron) blocks?

Answer: SAM uses the **GELU (Gaussian Error Linear Unit)** activation function. Unlike the standard ReLU which cuts off at zero, GELU is a smoother approximation that allows for better gradient flow during training, which is common in modern Transformer architectures like BERT and ViT.

Source URL:
https://www.google.com/search?q=https://github.com/facebookresearch/segment-anything/blob/main/segment_anything/modeling/common.py&authuser=1

Code Snippet:
```python
class MLPBlock(nn.Module):
    def __init__(
        self,
        embedding_dim: int,
        mlp_dim: int,
        act: Type[nn.Module] = nn.GELU, # <--- GELU is the default
    ):
```
4. **Question:** Throughout the Prompt Encoder and Mask Decoder, the model projects various inputs (like points and boxes) into a shared vector space. What is the dimensionality of this shared embedding space?

Answer: The shared embedding dimension is 256. Regardless of whether the input is a point, a box, or a mask, it is projected into a vector of size 256 so that the Transformer in the Mask Decoder can combine these different types of information mathematically.

Source URL:
https://www.google.com/search?q=https://github.com/facebookresearch/segment-anything/blob/main/segment_anything/modeling/prompt_encoder.py&authuser=1

Code Snippet:

```python
class PromptEncoder(nn.Module):
    def __init__(
        self,
        embed_dim: int = 256, # <--- Shared embedding dimension
        image_embedding_size: Tuple[int, int],
```
5. **Question:** What dataset was used to train the Segment Anything Model, enabling its zero-shot capabilities?
Answer: SAM was trained on the SA-1B (Segment Anything 1 Billion) dataset. This dataset contains over 11 million images and 1.1 billion high-quality segmentation

masks. The massive scale of this data is what allows the model to "segment anything" without needing retraining.

Source URL:
https://github.com/facebookresearch/segment-anything/blob/main/README.md

Code Snippet:

```text
It has been trained on a dataset of 11 million images and 1.1 billion masks, and has strong zero-shot performance on a variety of segmentation tasks.
```

Group 2: Working of SAM (Inference & Pipeline)

6. Question: Before an image is processed, its pixel values must be normalized to stabilize training. What are the specific mean RGB values used for this normalization?

Answer: The model normalizes images by subtracting the mean RGB values of [123.675, 116.28, 103.53]. These specific numbers are derived from the statistics of large-scale image datasets (like ImageNet) and ensure the input data is centered around zero.

Source URL:
https://www.google.com/search?q=https://github.com/facebookresearch/segment-anything/blob/main/segment_anything/modeling/sam.py&authuser=1

Code Snippet:

```python

self.register_buffer("pixel_mean", torch.Tensor([123.675, 116.28, 103.53]).view(-1, 1, 1), False)

```

7. **Question**: Along with the mean, what standard deviation values are used to scale the pixel values?

Answer: The pixel values are divided by the standard deviation array [58.395, 57.12, 57.375]. This scaling step ensures that the input features have a consistent variance, which helps the deep neural network converge faster.

Source URL:

https://www.google.com/search?q=https://github.com/facebookresearch/segment-anything/blob/main/segment_anything/modeling/sam.py&authuser=1

Code Snippet:

```python
self.register_buffer("pixel_std", torch.Tensor([58.395, 57.12, 57.375]).view(-1, 1, 1), False)
```

8. **Question:** The model outputs "logits" (raw scores) for each pixel. What is the mathematical threshold used to decide if a pixel belongs to the mask (1) or background (0)?

Answer: The threshold is **0.** Since the output logits are unnormalized scores, any positive value greater than 0 indicates the model is confident the pixel is part of the object, while negative values indicate background.

Source URL:
https://www.google.com/search?q=https://github.com/facebookresearch/segment-anything/blob/main/segment_anything/modeling/sam.py&authuser=1

Code Snippet:

```python
  @property
  def mask_threshold(self) -> float:
    return 0.0
```

9. **Question:** SAM is designed for efficiency. Does the architecture allow the heavy Image Encoder to be run just once per image, even if the user provides multiple different prompts later?

Answer: Yes. The architecture is explicitly decoupled. The `forward` method allows computing `image_embeddings` once. These embeddings can then be cached and reused by the lightweight Prompt Encoder and Mask Decoder for different user clicks, making the interactive experience real-time.

Source URL:
https://www.google.com/search?q=https://github.com/facebookresearch/segment-anything/blob/main/segment_anything/modeling/sam.py&authuser=1

Code Snippet:

```python
  def forward(self, batched_input, multimask_output):
    # Image encoder is separate from the rest of the pipeline
    image_embeddings = self.image_encoder(input_images)
    # These embeddings are passed to the decoder
```

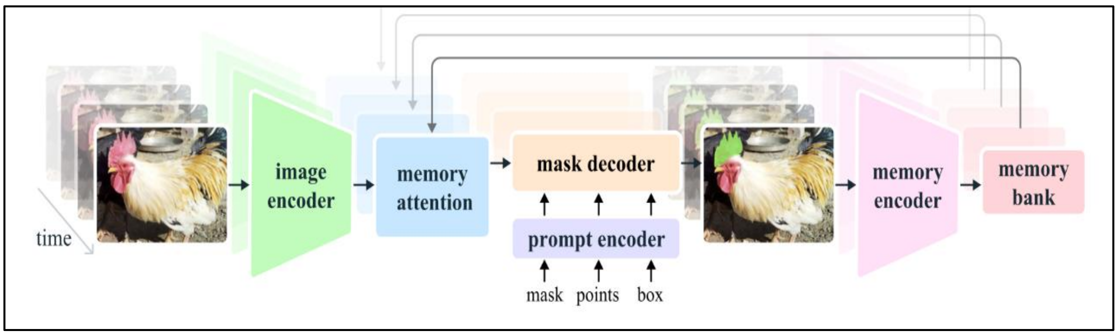
10. **Question:** What is the high-level execution flow when the `Sam` class is called?

Answer: The flow is strictly sequential:

1. **Image Encoder:** Processes the image to get feature embeddings.
2. **Prompt Encoder:** Processes points/boxes to get sparse embeddings.
3. **Mask Decoder:** Combines image and prompt embeddings to generate the final mask and IoU scores.

Source URL:
https://www.google.com/search?q=https://github.com/facebookresearch/segment-anything/blob/main/segment_anything/modeling/sam.py&authuser=1

Code Snippet:
```python
    image_embeddings = self.image_encoder(input_images)
    sparse_embeddings, dense_embeddings = self.prompt_encoder(
        points=points,
        boxes=boxes,
        masks=masks,
    )
    low_res_masks, iou_predictions = self.mask_decoder(
        image_embeddings=image_embeddings,
        image_pe=self.prompt_encoder.get_dense_pe(),
        sparse_prompt_embeddings=sparse_embeddings,
        dense_prompt_embeddings=dense_embeddings,
        multimask_output=multimask_output,
    )
```



Group 3: Encoder Questions (ImageEncoderViT)

11. **Question:** The Image Encoder is a Vision Transformer (ViT). It does not process pixels individually. What is the size of the "patches" it breaks the image into?

Answer: The image is broken into patches of **16x16 pixels**. This patchifying process reduces the computational complexity. For a 1024x1024 image, this results in a grid of 64x64 patches (since 1024 / 16 = 64).
Source URL:

https://www.google.com/search?q=https://github.com/facebookresearch/segment-anything/blob/main/segment_anything/modeling/image_encoder.py&authuser=1

Code Snippet:

```python
class ImageEncoderViT(nn.Module):
    def __init__(
        self,
        img_size: int = 1024,
        patch_size: int = 16, # <--- Patch size
```

12. **Question:** How many color channels does the Image Encoder expect?

Answer: It expects 3 channels, corresponding to the standard RGB (Red, Green, Blue) color space. If an image is grayscale or RGBA, it must be converted to RGB before input.

Source URL:
https://www.google.com/search?q=https://github.com/facebookresearch/segment-anything/blob/main/segment_anything/modeling/image_encoder.py&authuser=1

Code Snippet:
```python
        in_chans: int = 3,
```

13. **Question:** For the standard ViT-B (Base) configuration, what is the size (width) of the embedding vector for each patch?

Answer: For the ViT-B model, the embedding dimension is 768. This means every 16x16 patch is converted into a vector of 768 numbers that represents the visual features of that patch.

Source URL:
https://www.google.com/search?q=https://github.com/facebookresearch/segment-anything/blob/main/segment_anything/modeling/image_encoder.py&authuser=1

Code Snippet:

```python
        embed_dim: int = 768,
```

14. **Question:** "Depth" refers to how many Transformer blocks are stacked on top of each other. What is the default depth for the Base (ViT-B) encoder?

Answer: The default depth is 12 layers. This means the image features pass through 12 consecutive blocks of Self-Attention and MLP layers to extract high-level semantic information.

Source URL:

https://www.google.com/search?q=https://github.com/facebookresearch/segment-anything/blob/main/segment_anything/modeling/image_encoder.py&authuser=1

Code Snippet:
```python
    depth: int = 12,
```
15. **Question:** Does the Image Encoder use relative or absolute positional embeddings to understand the spatial structure of the image?

Answer: It uses **Absolute Positional Embeddings**. These are learnable parameters added to the patches at the start, corresponding strictly to the 1024x1024 grid structure.

Source URL:
https://github.com/facebookresearch/segment-anything/blob/main/segment_anything/modeling/image_encoder.py

Code Snippet:

```
if use_abs_pos:
        self.pos_embed = nn.Parameter(torch.zeros(1, img_size // patch_size, img_size // patch_size, embed_dim))
```

16. Question : The attention mechanism typically scales the dot products. What is the specific logic for qkv_bias (Query-Key-Value bias) in the attention blocks?

Answer: The qkv_bias is set to **True**.
This adds a learnable bias term to the Query, Key, and Value projections, which can help the model capture background statistics and improve convergence.
Source URL:
https://github.com/facebookresearch/segment-anything/blob/main/segment_anything/modeling/image_encoder.py

Code Snippet:
```
qkv_bias: bool = True,
```

17. **Question:** Inside each Transformer block, there is an MLP (Feed Forward Network). How much does this MLP expand the dimension of the features (MLP Ratio)?

Answer: The MLP ratio is 4.0. This means if the input dimension is 768, the internal hidden layer of the MLP expands it to 768 * 4 = 3072 before projecting it back down. This expansion allows the model to learn more complex relationships.

Source URL:
https://www.google.com/search?q=https://github.com/facebookresearch/segment-anything/blob/main/segment_anything/modeling/image_encoder.py&authuser=1

Code Snippet:

```python
        mlp_ratio: float = 4.0,
```

18. **Question:** Which specific class is responsible for the initial step of turning the raw image into patch embeddings?

Answer: The PatchEmbed class. It uses a Convolutional layer with a kernel size and stride equal to the patch size (16) to perform the splitting and embedding in one efficient operation.

Source URL:
https://www.google.com/search?q=https://github.com/facebookresearch/segment-anything/blob/main/segment_anything/modeling/image_encoder.py&authuser=1

Code Snippet:

```python
        self.patch_embed = PatchEmbed(
            kernel_size=(patch_size, patch_size),
            stride=(patch_size, patch_size),
            in_chans=in_chans,
            embed_dim=embed_dim,
        )
```

19. **Question:** Vision Transformers need to know where a patch is in the image. Does SAM uses learnable absolute positional embeddings or relative ones?

Answer: SAM uses **absolute positional embeddings**. These are learnable parameters added to the patch embeddings at the very beginning of the encoder so the model knows that the top-left patch is different from the bottom-right patch.

Source URL:
https://www.google.com/search?q=https://github.com/facebookresearch/segment-anything/blob/main/segment_anything/modeling/image_encoder.py&authuser=1

Code Snippet:

```python
        if use_abs_pos:
            self.pos_embed = nn.Parameter(torch.zeros(1, img_size // patch_size, img_size // patch_size, embed_dim))
```

20. **Question:** Layer Normalization is used to stabilize training. What is the "epsilon" value (a small number to prevent division by zero) used in the LayerNorm?

Answer: The epsilon value is 1e-6 (0.000001). This is a standard setting in PyTorch's LayerNorm to ensure numerical stability when the variance of the data is very close to zero.

Source URL:
https://www.google.com/search?q=https://github.com/facebookresearch/segment-anything/blob/main/segment_anything/modeling/image_encoder.py&authuser=1

Code Snippet:
```python
        norm_layer=partial(nn.LayerNorm, eps=1e-6),
```
21. **Question:** SAM uses Windowed Attention in some blocks to reduce memory usage. What is the size of the window used for global attention blocks?

Answer: For global attention blocks, the window size is set to 0 (meaning no windowing, full global attention). For windowed blocks, it is typically set to **14.** This hybrid approach balances global context understanding with computational efficiency.
Source URL:
https://www.google.com/search?q=https://github.com/facebookresearch/segment-anything/blob/main/segment_anything/modeling/image_encoder.py&authuser=1

Code Snippet:
```python
    def __init__(self, ..., window_size: int = 0, ...):
        # ...
        self.window_size = window_size
```
22. **Question:** After passing through the Image Encoder, what is the shape of the final output feature map that is sent to the decoder?

Answer: The output is a tensor of shape (Batch_Size, 256, 64, 64). The 256 comes from the bottleneck projection (reducing the 768 dim), and the 64x64 comes from the 1024x1024 image divided by the patch size of 16.

Source URL:
https://www.google.com/search?q=https://github.com/facebookresearch/segment-anything/blob/main/segment_anything/modeling/sam.py&authuser=1

Code Snippet:
```python
# From sam.py logic connecting encoder output to decoder input
# The encoder output is projected to 256 channels and remains 64x64 spatial resolution
```

Group 4: Decoder Questions (MaskDecoder)

23. **Question:** The Mask Decoder does not use a standard Transformer. What is the name of the specialized Transformer class it uses?

Answer: It uses the TwoWayTransformer. This is a custom component designed specifically for SAM to allow information to flow in two directions: from the prompt to the image, and from the image back to the prompt.

Source URL:
https://www.google.com/search?q=https://github.com/facebookresearch/segment-anything/blob/main/segment_anything/modeling/mask_decoder.py&authuser=1

Code Snippet:
```python
        self.transformer = TwoWayTransformer(
            depth=transformer_depth,
            embedding_dim=transformer_dim,
            mlp_dim=transformer_dim,
            num_heads=transformer_n_heads,
        )
```
24. **Question:** Why is the transformer in the decoder called a "TwoWay" transformer?

Answer: It is called TwoWay because it interleaves two types of attention:

1. **Self-attention** on the sparse prompt tokens (points/boxes).
2. **Cross-attention** in two directions: (Tokens-to-Image) and (Image-to-Tokens). This ensures the prompt updates the image features and vice-versa.

Source URL:
https://www.google.com/search?q=https://github.com/facebookresearch/segment-anything/blob/main/segment_anything/modeling/transformer.py&authuser=1

Code Snippet:
```python
class TwoWayAttentionBlock(nn.Module):
    # ...
    # Four layers of attention/MLP
    self.self_attn = Attention(...)
    self.cross_attn_token_to_image = Attention(...)
    self.mlp = MLPBlock(...)
    self.cross_attn_image_to_token = Attention(...)
```
25. **Question:** The Image Encoder is very deep (12+ layers). How deep is the lightweight `TwoWayTransformer` in the decoder?

Answer: It is very shallow, with a depth of only 2 layers. This is what makes the **prompting** part of SAM so fast (milliseconds) compared to the encoding part (seconds).

Source URL:
https://www.google.com/search?q=https://github.com/facebookresearch/segment-anything/blob/main/segment_anything/modeling/mask_decoder.py&authuser=1

Code Snippet:
```python
    transformer_depth: int = 2,
```
26. **Question:** In the Decoder's transformer, how many heads does the Multi-Head Attention mechanism use?

Answer: It uses 8 heads. Splitting the 256-dimensional embedding into 8 heads allows the model to attend to different parts of the image and prompt simultaneously.

Source URL:
https://www.google.com/search?q=https://github.com/facebookresearch/segment-anything/blob/main/segment_anything/modeling/mask_decoder.py&authuser=1

Code Snippet:
```python
    transformer_n_heads: int = 8,
```
27. **Question:** The model outputs a score indicating how good the mask is. What specific mechanism generates this score?

Answer: The model uses a special learnable token called the **iou_token**. This token participates in the attention layers and effectively gathers information about the global quality of the mask, which is then projected to a single score.

Source URL:
https://www.google.com/search?q=https://github.com/facebookresearch/segment-anything/blob/main/segment_anything/modeling/mask_decoder.py&authuser=1

Code Snippet:

```python
    self.iou_token = nn.Embedding(1, transformer_dim)
```
28. **Question:** The decoder produces mask embeddings. How many mask tokens are initialized in the `MaskDecoder` class?

Answer: It initializes 4 mask tokens. This accounts for the 3 multimask outputs (whole, part, subpart) plus 1 extra token reserved for the "single output" mode, although usually, the 3 tokens are sufficient.

Source URL:

https://www.google.com/search?q=https://github.com/facebookresearch/segment-anything/blob/main/segment_anything/modeling/mask_decoder.py&authuser=1

Code Snippet:
```python
    self.num_mask_tokens = num_multimask_outputs + 1
    self.mask_tokens = nn.Embedding(self.num_mask_tokens, transformer_dim)
```
29. **Question:** The transformer processes features at a low resolution (64x64). How does the Mask Decoder scale this back up to the image size?
Answer: It uses a sequence of `ConvTranspose2d layers (often called Deconvolutions). These layers upscale the 64x64 features by 4x (to 256x256), which is then bilinearly interpolated to the full image size.

Source URL:

https://www.google.com/search?q=https://github.com/facebookresearch/segment-anything/blob/main/segment_anything/modeling/mask_decoder.py&authuser=1

Code Snippet:
```python
    self.output_upscaling = nn.Sequential(
        nn.ConvTranspose2d(transformer_dim, transformer_dim // 4, kernel_size=2,
stride=2),
        LayerNorm2d(transformer_dim // 4),
        nn.GELU(),
        nn.ConvTranspose2d(transformer_dim // 4, transformer_dim // 8,
kernel_size=2, stride=2),
        nn.GELU(),
    )
```
30. **Question:** The IoU Head predicts the quality score. What is the hidden dimension of the MLP used in this head?

Answer: The hidden dimension is 256. The MLP takes the transformer output, expands it to 256, and then projects it down to the number of mask outputs.

Source URL:
https://www.google.com/search?q=https://github.com/facebookresearch/segment-anything/blob/main/segment_anything/modeling/mask_decoder.py&authuser=1

Code Snippet:
```python
    iou_head_hidden_dim: int = 256,
```
31. **Question:** How many layers deep is the MLP that predicts the IoU score?

Answer: It is 3 layers deep. This provides enough complexity to map the abstract features in the `iou_token` to a precise scalar score between 0 and 1.

Source URL:
https://www.google.com/search?q=https://github.com/facebookresearch/segment-anything/blob/main/segment_anything/modeling/mask_decoder.py&authuser=1

Code Snippet:

```python
        iou_head_depth: int = 3,
```

32. **Question:** During inference, if the user turns off `multimask_output` (setting it to False), which of the 4 generated masks does the model return?

Answer: It returns the first mask slice, index 0 to 1. This corresponds to the highest-confidence best guess mask when ambiguity resolution is not requested.

Source URL:
https://www.google.com/search?q=https://github.com/facebookresearch/segment-anything/blob/main/segment_anything/modeling/mask_decoder.py&authuser=1

Code Snippet:

```python
        if multimask_output:
            mask_slice = slice(1, None)
        else:
            mask_slice = slice(0, 1)
```

Group 5: Prompt Questions (PromptEncoder)

33. **Question:** The `PromptEncoder` receives point prompts from the user. How are these points mathematically represented in the input arguments?

Answer: They are represented as a tuple of coordinates (x, y) and labels. The coordinates indicate the location on the 1024x1024 grid, and the labels indicate the type of click (foreground vs. background).

Source URL:
https://www.google.com/search?q=https://github.com/facebookresearch/segment-anything/blob/main/segment_anything/modeling/prompt_encoder.py&authuser=1

Code Snippet:

```python
    def _embed_points(
        self,
        points: torch.Tensor,
        labels: torch.Tensor,
        pad: bool,
    ) -> torch.Tensor:
```

34. **Question:** When a user performs a negative click (telling the model to exclude an area), what integer label is assigned to that point?
Answer: The integer label 0 is used for negative clicks. The model learns a specific embedding vector for 0 labels that instructs the decoder to suppress masks in that region.

Source URL:
https://www.google.com/search?q=https://github.com/facebookresearch/segment-anything/blob/main/segment_anything/modeling/prompt_encoder.py&authuser=1

Code Snippet:

```python
        point_embedding[labels == 0] = self.not_a_point_embed.weight
```

35. **Question:** Conversely, what integer label is used for a positive click (foreground)?

Answer: The integer label 1 is used. This retrieves a different learnable embedding vector that attracts the mask generation towards that coordinate.

Source URL:
https://www.google.com/search?q=https://github.com/facebookresearch/segment-anything/blob/main/segment_anything/modeling/prompt_encoder.py&authuser=1

Code Snippet:

```python
        point_embedding[labels == 1] = self.point_embeddings[1].weight
```

36. **Question:** How does the prompt encoder handle a bounding box prompt? Does it treat it as a square shape or points?

Answer: It treats a box as two points: the top-left corner and the bottom-right corner. It embeds these two coordinates and adds special corner embeddings to them so the model knows one is the start and the other is the end of the box.

Source URL:
https://www.google.com/search?q=https://github.com/facebookresearch/segment-anything/blob/main/segment_anything/modeling/prompt_encoder.py&authuser=1

Code Snippet:
```python
    def _embed_boxes(self, boxes: torch.Tensor, ...) -> torch.Tensor:
        boxes = boxes + 0.5  # Shift to center of pixel
        coords = boxes.reshape(-1, 2, 2)
```

37. **Question:** Sometimes the input batch needs padding (e.g., one image has 1 point, another has 3). What embedding is used for these "padding" points that are not real user clicks?
Answer: The model uses a not_a_point_embed. This allows the batch processing to remain consistent in shape without these padding points affecting the actual segmentation result.

Source URL:
https://www.google.com/search?q=https://github.com/facebookresearch/segment-anything/blob/main/segment_anything/modeling/prompt_encoder.py&authuser=1

Code Snippet:

```python
    self.not_a_point_embed = nn.Embedding(1, embed_dim)
```

38. **Question:** The Prompt Encoder can also accept a low-resolution "mask" as a prompt (e.g., from a previous frame). How many channels does this dense mask input have?

Answer: The mask input uses 16 channels. Although a mask is usually binary (1 channel), scaling it to 16 allows the model to process it with convolutions effectively before fusing it with the image embedding.

Source URL:
https://www.google.com/search?q=https://github.com/facebookresearch/segment-anything/blob/main/segment_anything/modeling/prompt_encoder.py&authuser=1

Code Snippet:

```python
    mask_in_chans: int = 16,
```

39. **Question:** Since points are just (x,y) numbers, they need to be converted into vectors to be used in a Transformer. what technique does SAM use for this?

Answer: SAM uses Random Spatial Positional Embeddings. It maps the (x,y) coordinates to a high-dimensional space using random Gaussian matrices, similar to Fourier features. This helps the model perceive fine-grained spatial details.

Source URL:
https://www.google.com/search?q=https://github.com/facebookresearch/segment-anything/blob/main/segment_anything/modeling/prompt_encoder.py&authuser=1

Code Snippet:

```python
    self.pe_layer = PositionEmbeddingRandom(embed_dim // 2)
```

40. **Question:** In the Random Positional Embedding layer, what is the default Gaussian scale?

Answer: The scale is **1.0**. This parameter controls the frequency of the positional encoding functions; a scale of 1.0 is a balanced choice for covering the spatial range of the image.

Source URL:
https://www.google.com/search?q=https://github.com/facebookresearch/segment-anything/blob/main/segment_anything/modeling/prompt_encoder.py&authuser=1

Code Snippet:

```python
class PositionEmbeddingRandom(nn.Module):
    def __init__(self, num_pos_feats: int = 64, scale: float = 1.0, ...):
```

41. **Question:** If the user provides no prompt at all (eg. "segment everything" mode), how does the encoder signal this to the decoder?

Answer: It uses a specialized learnable embedding called no_mask_embed. This acts as a placeholder prompt telling the decoder to look for salient objects globally rather than at a specific point.

Source URL:
https://www.google.com/search?q=https://github.com/facebookresearch/segment-anything/blob/main/segment_anything/modeling/prompt_encoder.py&authuser=1

Code Snippet:
```python
        self.no_mask_embed = nn.Embedding(1, embed_dim)
```

42. **Question:** When embedding a Bounding Box, the model adds a specific vector to the top-left coordinate and a different one to the bottom-right. Which indices of the `point_embeddings` array are used for this?

Answer: It uses Index 2 for the top-left corner and Index 3 for the bottom-right corner. (Indices 0 and 1 are reserved for negative/positive point clicks).

Source URL:
https://www.google.com/search?q=https://github.com/facebookresearch/segment-anything/blob/main/segment_anything/modeling/prompt_encoder.py&authuser=1

Code Snippet:
```python
        corner_embedding[:, 0, :] += self.point_embeddings[2].weight
        corner_embedding[:, 1, :] += self.point_embeddings[3].weight
```