**1. What are the pros and cons of using a stateful RNN versus a stateless RNN?**

* **Stateful RNN**:
  + **Pros**:
    - Maintains state across batches, which is useful for long sequences or when the sequence order matters.
    - Can model long-term dependencies more effectively.
  + **Cons**:
    - Requires careful handling of batch sizes and sequence lengths.
    - More difficult to implement and debug.
* **Stateless RNN**:
  + **Pros**:
    - Easier to implement and debug.
    - More flexible with batch sizes and sequence lengths.
  + **Cons**:
    - Loses state between batches, making it less effective for long-term dependencies.

**2. Why do people use Encoder-Decoder RNNs rather than plain sequence-to-sequence RNNs for automatic translation?**

* **Encoder-Decoder RNNs**:
  + **Advantages**:
    - The encoder processes the input sequence into a fixed-size context vector, which captures the entire input sequence.
    - The decoder generates the output sequence step-by-step, conditioned on the context vector.
    - This architecture is more effective for handling variable-length input and output sequences.
* **Plain sequence-to-sequence RNNs**:
  + Struggle with long sequences due to the vanishing gradient problem.
  + Less effective at capturing the entire input sequence in a single context vector.

**3. How can you deal with variable-length input sequences? What about variable-length output sequences?**

* **Variable-length input sequences**:
  + Use **padding** to ensure all sequences in a batch have the same length.
  + Use **masking** to ignore padded values during training.
* **Variable-length output sequences**:
  + Use an **Encoder-Decoder architecture** with a dynamic output sequence length.
  + Use **beam search** or **greedy search** to generate sequences of varying lengths.

**4. What is beam search and why would you use it? What tool can you use to implement it?**

* **Beam search**:
  + A heuristic search algorithm that explores multiple possible sequences at each step, keeping the top k (beam width) candidates.
* **Why use it**:
  + Improves the quality of generated sequences (e.g., in machine translation or text generation) by considering multiple hypotheses.
* **Tool**:
  + TensorFlow's tf.nn.ctc\_beam\_search\_decoder or libraries like Hugging Face's transformers provide beam search implementations.

**5. What is an attention mechanism? How does it help?**

* **Attention mechanism**:
  + A technique that allows the model to focus on specific parts of the input sequence when generating each output token.
* **How it helps**:
  + Improves performance on tasks like machine translation by allowing the model to dynamically weigh the importance of different input tokens.
  + Helps handle long sequences by reducing the reliance on a fixed-size context vector.

**6. What is the most important layer in the Transformer architecture? What is its purpose?**

* **Most important layer**: **Self-attention layer**.
* **Purpose**:
  + Computes attention scores between all tokens in the input sequence, allowing the model to capture dependencies regardless of distance.
  + Enables parallel processing and improves performance on tasks like machine translation and text generation.

**7. When would you need to use sampled softmax?**

* **Sampled softmax**:
  + A computationally efficient alternative to the full softmax, used when the output vocabulary is very large (e.g., in language modeling or machine translation).
* **When to use it**:
  + When training on large datasets with a large vocabulary, as it reduces the computational cost of computing the softmax over all possible outputs.