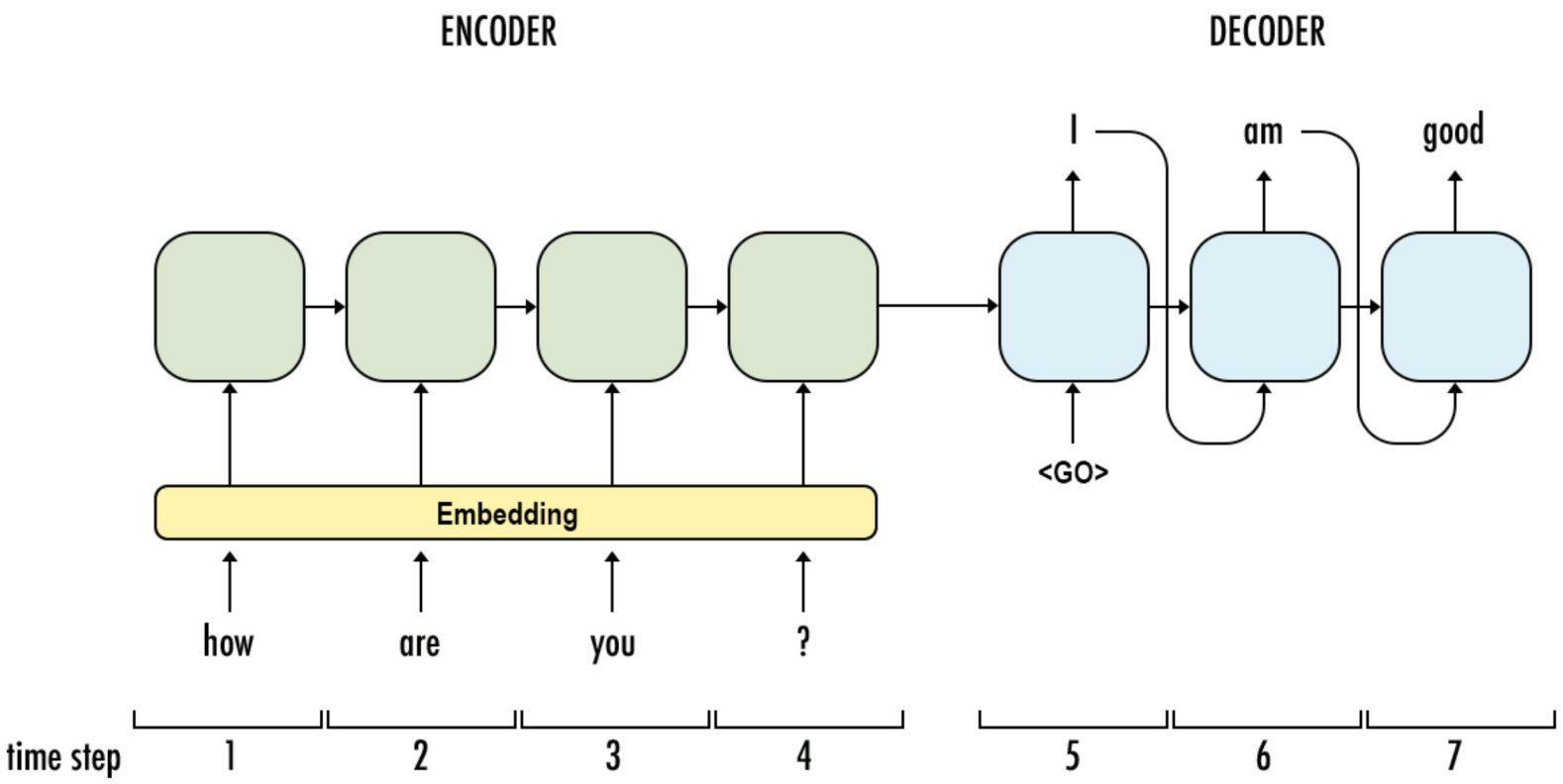
In [21]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

Base Seq2Seq Model

Seq2Seq models are constructed using key components like Long Short-Term Memory (LSTM) units, which are specialized types of recurrent neural networks (RNNs) that effectively capture temporal dependencies in data. LSTMs help in processing sequences by remembering important information over longer periods and forgetting irrelevant details, making them useful for handling tasks involving sequences of varying lengths.



I found the following notebook https://www.kaggle.com/code/harshjain123/machine-translation-seq2seq-lstms by Harsh Jain very insightful to understand this process, its will be a great place to understand the data processing pipeline.

In [6]:
```
@keras.saving.register_keras_serializable(package="Custom", name="S2S")
class S2S(Model):
    def __init__(self, in_vocab, out_vocab, in_timesteps, out_timesteps, units, **kwargs):
        super(S2S, self).__init__(**kwargs)

        self.in_vocab = in_vocab
        self.out_vocab = out_vocab
        self.in_timesteps = in_timesteps
        self.out_timesteps = out_timesteps
        self.units = units

        # Define the layers
        self.embed = Embedding(input_dim=in_vocab, output_dim=units, mask_zero=True)
        self.encoder_lstm = LSTM(units)
        self.r_vector = RepeatVector(out_timesteps)
        self.decoder_lstm = LSTM(units, return_sequences=True)
        self.dense = Dense(out_vocab, activation='softmax')

    def call(self, inputs):
        # Define the forward pass
        x = self.embed(inputs)                          # (batch size, in_timesteps, units)
        x = self.encoder_lstm(x)                        # (batch size, units)
        x = self.r_vector(x)                            # (batch size, out_timesteps, units)
        x = self.decoder_lstm(x)                        # (batch size, out_timesteps, units)
        output = self.dense(x)                          # (batch size, out_timesteps, out_vocab)
        return output

    def get_config(self):
        config = super(S2S, self).get_config()
        config.update({
              'in_vocab': self.in_vocab,
              'out_vocab': self.out_vocab,
              'in_timesteps': self.in_timesteps,
```