Import Libraries

In [17]:
```
import tensorflow as tf
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from tensorflow.keras.layers import TextVectorization
import re,string
from tensorflow.keras.layers import LSTM,Dense,Embedding,Dropout,LayerNorma
```

In [18]:
```
df=pd.read_csv('Desktop\My project\dialogs.txt',sep='\t',names=['question',
print(f'Dataframe size: {len(df)}')
df.head(3725)
```

```
Dataframe size: 3725
```

Out[18]:

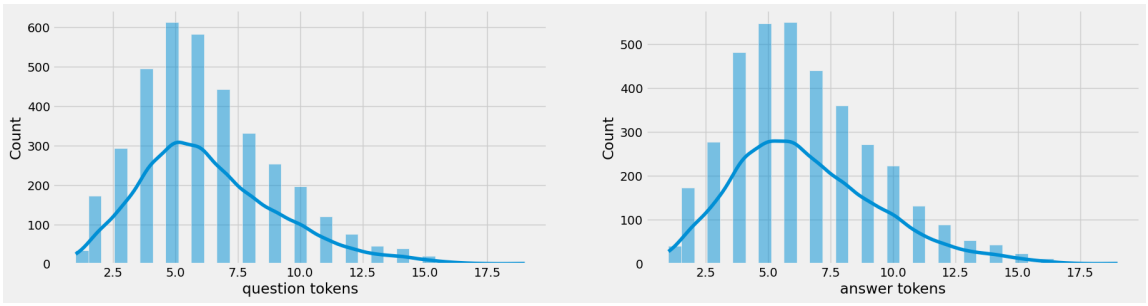
	question	answer
0	hi, how are you doing?	i'm fine. how about yourself?
1	i'm fine. how about yourself?	i'm pretty good. thanks for asking.
2	i'm pretty good. thanks for asking.	no problem. so how have you been?
3	no problem. so how have you been?	i've been great. what about you?
4	i've been great. what about you?	i've been good. i'm in school right now.
...	...	...
3720	that's a good question. maybe it's not old age.	are you right-handed?
3721	are you right-handed?	yes. all my life.
3722	yes. all my life.	you're wearing out your right hand. stop using...
3723	you're wearing out your right hand. stop	but i do all my writing with my right hand.

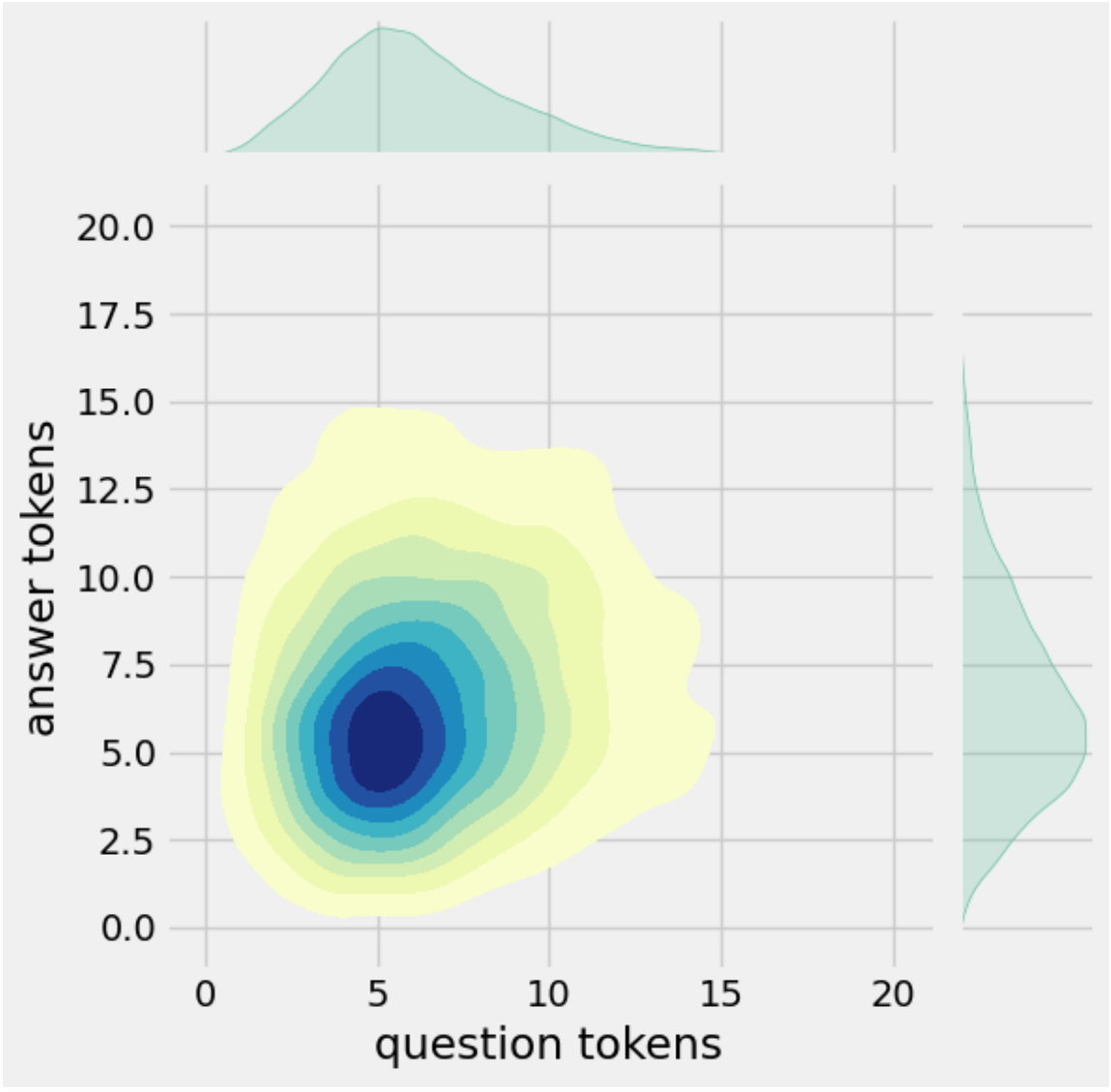
Data Preprocessing

Data Visualization

In [19]:

```
df['question tokens']=df['question'].apply(lambda x:len(x.split()))
df['answer tokens']=df['answer'].apply(lambda x:len(x.split()))
plt.style.use('fivethirtyeight')
fig,ax=plt.subplots(nrows=1,ncols=2,figsize=(20,5))
sns.set_palette('Set2')
sns.histplot(x=df['question tokens'],data=df,kde=True,ax=ax[0])
sns.histplot(x=df['answer tokens'],data=df,kde=True,ax=ax[1])
sns.jointplot(x='question tokens',y='answer tokens',data=df,kind='kde',fill
plt.show()
```





Text Cleaning

In [20]:

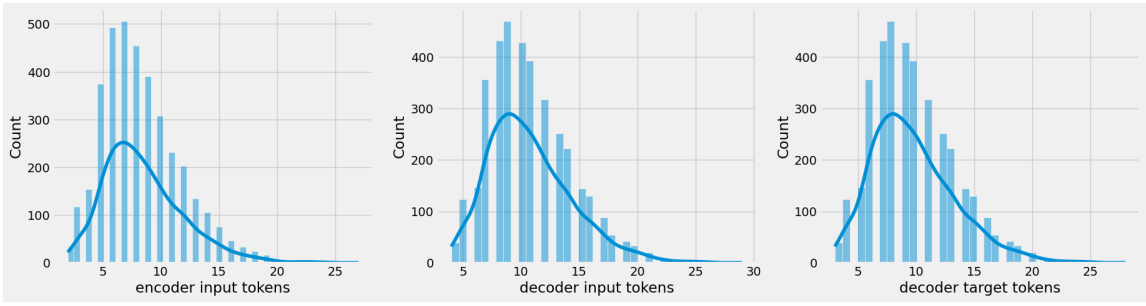
```
def clean_text(text):
    text=re.sub('-',' ',text.lower())
    text=re.sub('[.]',' . ',text)
    text=re.sub('[1]',' 1 ',text)
    text=re.sub('[2]',' 2 ',text)
    text=re.sub('[3]',' 3 ',text)
    text=re.sub('[4]',' 4 ',text)
    text=re.sub('[5]',' 5 ',text)
    text=re.sub('[6]',' 6 ',text)
    text=re.sub('[7]',' 7 ',text)
    text=re.sub('[8]',' 8 ',text)
    text=re.sub('[9]',' 9 ',text)
    text=re.sub('[0]',' 0 ',text)
    text=re.sub('[,]',' , ',text)
    text=re.sub('[?]',' ? ',text)
    text=re.sub('[!]',' ! ',text)
    text=re.sub('[$]',' $ ',text)
    text=re.sub('[&]',' & ',text)
    text=re.sub('[/]',' / ',text)
    text=re.sub('[:]',' : ',text)
    text=re.sub('[;]',' ; ',text)
    text=re.sub('[*]',' * ',text)
    text=re.sub('[\']',' \' ',text)
    text=re.sub('[\"]',' \" ',text)
    text=re.sub('\t',' ',text)
    return text

df.drop(columns=['answer tokens','question tokens'],axis=1,inplace=True)
df['encoder_inputs']=df['question'].apply(clean_text)
df['decoder_targets']=df['answer'].apply(clean_text)+' <end>'
df['decoder_inputs']='<start> '+df['answer'].apply(clean_text)+' <end>'

df.head(3725)
```

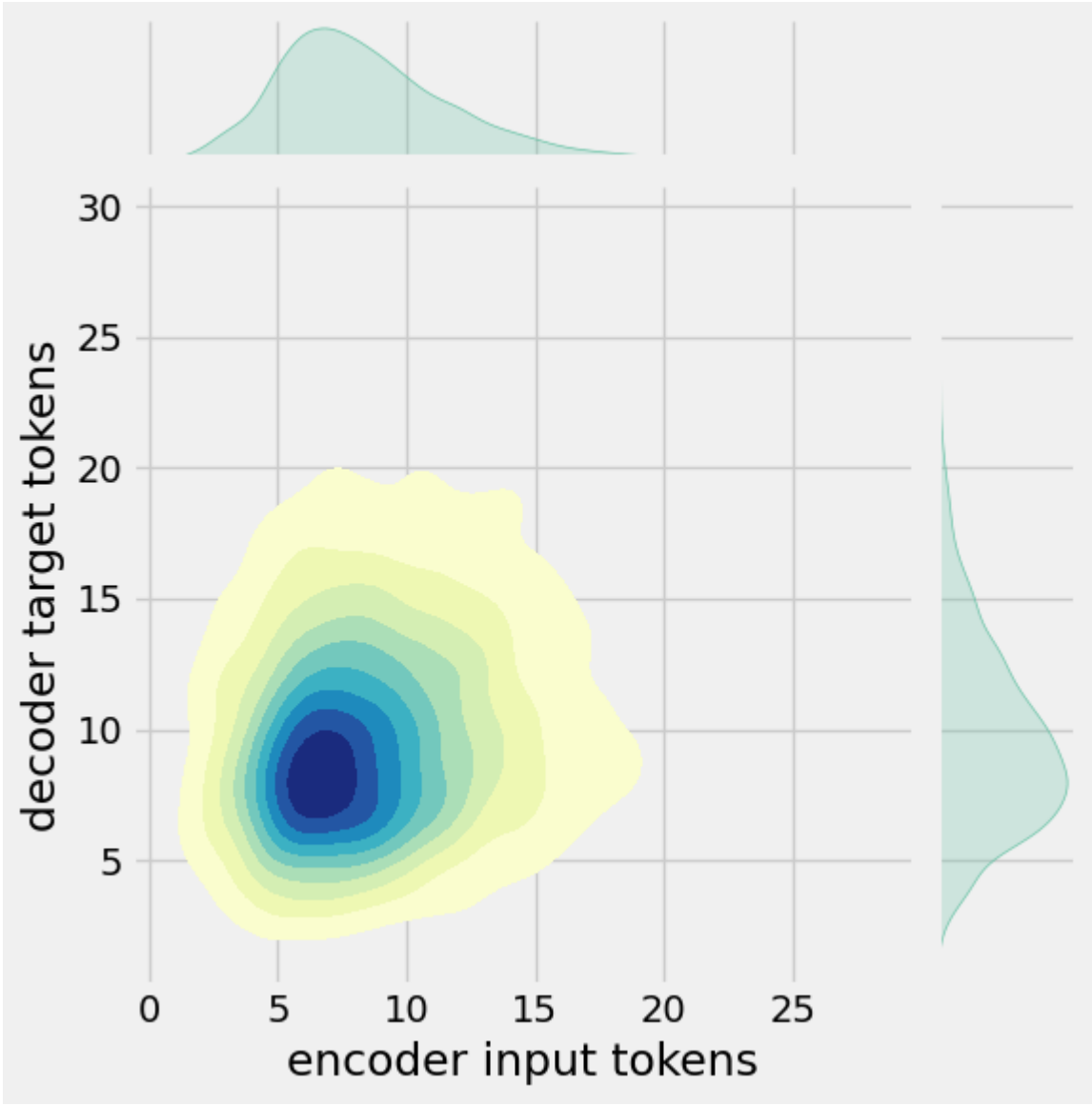
Out[20]:

	question	answer	encoder_inputs	decoder_targets	decoder_inputs
0	hi, how are you doing?	i'm fine. how about yourself?	hi , how are you doing ?	i ' m fine . how about yourself ? <end>	<start> i ' m fine . how about yourself ? <end>
1	i'm fine. how about yourself?	i'm pretty good. thanks for asking.	i ' m fine . how about yourself ?	i ' m pretty good . thanks for asking . <end>	<start> i ' m pretty good . thanks for asking...
2	i'm pretty good. thanks for asking.	no problem. so how have you been?	i ' m pretty good . thanks for asking .	no problem . so how have you been ? <end>	<start> no problem . so how have you been ? ...
3	no problem. so how have you been?	i've been great. what about you?	no problem . so how have you been ?	i ' ve been great . what about you ? <end>	<start> i ' ve been great . what about you ? ...
4	i've been great. what about you?	i've been good. i'm in school right now.	i ' ve been great . what about you ?	i ' ve been good . i ' m in school right now ...	<start> i ' ve been good . i ' m in school ri...
...	...	...	...	...	...
3720	that's a good question. maybe it's not old age.	are you right-handed?	that ' s a good question . maybe it ' s not o...	are you right handed ? <end>	<start> are you right handed ? <end>
3721	are you right-handed?	yes. all my life.	are you right handed ?	yes . all my life . <end>	<start> yes . all my life . <end>
3722	yes. all my life.	you're wearing out your right hand. stop using...	yes . all my life .	you ' re wearing out your right hand . stop u...	<start> you ' re wearing out your right hand ....
3723	you're wearing out your right hand. stop using...	but i do all my writing with my right hand.	you ' re wearing out your right hand . stop u...	but i do all my writing with my right hand	<start> but i do all my writing with my right ...
3724	but i do all my writing with my right hand.	start typing instead. that way your left hand ...	but i do all my writing with my right hand .	start typing instead . that way your left han...	<start> start typing instead . that way your ...

3725 rows × 5 columns

In [21]:
```
df['encoder input tokens']=df['encoder_inputs'].apply(lambda x:len(x.split(
df['decoder input tokens']=df['decoder_inputs'].apply(lambda x:len(x.split(
df['decoder target tokens']=df['decoder_targets'].apply(lambda x:len(x.spli
plt.style.use('fivethirtyeight')
fig,ax=plt.subplots(nrows=1,ncols=3,figsize=(20,5))
sns.set_palette('Set2')
sns.histplot(x=df['encoder input tokens'],data=df,kde=True,ax=ax[0])
sns.histplot(x=df['decoder input tokens'],data=df,kde=True,ax=ax[1])
sns.histplot(x=df['decoder target tokens'],data=df,kde=True,ax=ax[2])
sns.jointplot(x='encoder input tokens',y='decoder target tokens',data=df,ki
plt.show()
```





In [22]:
```
print(f"After preprocessing: {' '.join(df[df['encoder input tokens'].max()=
print(f"Max encoder input length: {df['encoder input tokens'].max()}")
print(f"Max decoder input length: {df['decoder input tokens'].max()}")
print(f"Max decoder target length: {df['decoder target tokens'].max()}")

df.drop(columns=['question','answer','encoder input tokens','decoder input
params={
    "vocab_size":2500,
    "max_sequence_length":30,
    "learning_rate":0.008,
    "batch_size":149,
    "lstm_cells":256,
    "embedding_dim":256,
    "buffer_size":10000
}
learning_rate=params['learning_rate']
batch_size=params['batch_size']
embedding_dim=params['embedding_dim']
lstm_cells=params['lstm_cells']
vocab_size=params['vocab_size']
buffer_size=params['buffer_size']
max_sequence_length=params['max_sequence_length']
df.head(3725)
```

```
After preprocessing: for example ,  if your birth date is january  1  2  ,
1  9  8  7  ,  write  0  1  /  1  2  /  8  7  .
Max encoder input length: 27
Max decoder input length: 29
Max decoder target length: 28
```

Out[22]:

	encoder_inputs	decoder_targets	decoder_inputs
0	hi , how are you doing ?	i ' m fine . how about yourself ? <end>	<start> i ' m fine . how about yourself ? <end>
1	i ' m fine . how about yourself ?	i ' m pretty good . thanks for asking . <end>	<start> i ' m pretty good . thanks for asking...
2	i ' m pretty good . thanks for asking .	no problem . so how have you been ? <end>	<start> no problem . so how have you been ? ...
3	no problem . so how have you been ?	i ' ve been great . what about you ? <end>	<start> i ' ve been great . what about you ? ...
4	i ' ve been great . what about you ?	i ' ve been good . i ' m in school right now ...	<start> i ' ve been good . i ' m in school ri...
...	...	...	...
3720	that ' s a good question . maybe it ' s not o...	are you right handed ? <end>	<start> are you right handed ? <end>
3721	are you right handed ?	yes . all my life . <end>	<start> yes . all my life . <end>
3722	yes . all my life .	you ' re wearing out your right hand . stop u...	<start> you ' re wearing out your right hand ....
3723	you ' re wearing out your right hand . stop u...	but i do all my writing with my right hand	<start> but i do all my writing with my right ...
3724	but i do all my writing with my right hand .	start typing instead . that way your left han...	<start> start typing instead . that way your ...

3725 rows × 3 columns

Tokenization

In [23]:
```
vectorize_layer=TextVectorization(
    max_tokens=vocab_size,
    standardize=None,
    output_mode='int',
    output_sequence_length=max_sequence_length
)
vectorize_layer.adapt(df['encoder_inputs']+' '+df['decoder_targets']+' <sta
vocab_size=len(vectorize_layer.get_vocabulary())
print(f'Vocab size: {len(vectorize_layer.get_vocabulary())}')
print(f'{vectorize_layer.get_vocabulary()[:12]}')
```

```
Vocab size: 2443
['', '[UNK]', '<end>', '.', '<start>', "'", 'i', '?', 'you', ',', 'the',
'to']
```

In [24]:
```
def sequences2ids(sequence):
    return vectorize_layer(sequence)

def ids2sequences(ids):
    decode=''
    if type(ids)==int:
        ids=[ids]
    for id in ids:
        decode+=vectorize_layer.get_vocabulary()[id]+' '
    return decode

x=sequences2ids(df['encoder_inputs'])
yd=sequences2ids(df['decoder_inputs'])
y=sequences2ids(df['decoder_targets'])

print(f'Question sentence: hi , how are you ?')
print(f'Question to tokens: {sequences2ids("hi , how are you ?")[:10]}')
print(f'Encoder input shape: {x.shape}')
print(f'Decoder input shape: {yd.shape}')
print(f'Decoder target shape: {y.shape}')
```

```
Question sentence: hi , how are you ?
Question to tokens: [1971    9   45   24    8    7    0    0    0    0]
Encoder input shape: (3725, 30)
Decoder input shape: (3725, 30)
Decoder target shape: (3725, 30)
```

In [25]:
```
print(f'Encoder input: {x[0][:12]} ...')
print(f'Decoder input: {yd[0][:12]} ...')    # shifted by one time step of
print(f'Decoder target: {y[0][:12]} ...')
```

```
Encoder input: [1971    9   45   24    8  194    7    0    0    0    0
0] ...
Decoder input: [  4   6   5  38 646   3  45  41 563   7   2   0] ...
Decoder target: [  6   5  38 646   3  45  41 563   7   2   0   0] ...
```

In [26]:
```
data=tf.data.Dataset.from_tensor_slices((x,yd,y))
data=data.shuffle(buffer_size)

train_data=data.take(int(.9*len(data)))
train_data=train_data.cache()
train_data=train_data.shuffle(buffer_size)
train_data=train_data.batch(batch_size)
train_data=train_data.prefetch(tf.data.AUTOTUNE)
train_data_iterator=train_data.as_numpy_iterator()

val_data=data.skip(int(.9*len(data))).take(int(.1*len(data)))
val_data=val_data.batch(batch_size)
val_data=val_data.prefetch(tf.data.AUTOTUNE)

_=train_data_iterator.next()
print(f'Number of train batches: {len(train_data)}')
print(f'Number of training data: {len(train_data)*batch_size}')
print(f'Number of validation batches: {len(val_data)}')
print(f'Number of validation data: {len(val_data)*batch_size}')
print(f'Encoder Input shape (with batches): {_[0].shape}')
print(f'Decoder Input shape (with batches): {_[1].shape}')
print(f'Target Output shape (with batches): {_[2].shape}')
```

```
Number of train batches: 23
Number of training data: 3427
Number of validation batches: 3
Number of validation data: 447
Encoder Input shape (with batches): (149, 30)
Decoder Input shape (with batches): (149, 30)
Target Output shape (with batches): (149, 30)
```

In [ ]:

Build Models

Build Encoder

In [ ]:

In [27]:
```
class Encoder(tf.keras.models.Model):
    def __init__(self,units,embedding_dim,vocab_size,*args,**kwargs) -> Non
        super().__init__(*args,**kwargs)
        self.units=units
        self.vocab_size=vocab_size
        self.embedding_dim=embedding_dim
        self.embedding=Embedding(
            vocab_size,
            embedding_dim,
            name='encoder_embedding',
            mask_zero=True,
            embeddings_initializer=tf.keras.initializers.GlorotNormal()
        )
        self.normalize=LayerNormalization()
        self.lstm=LSTM(
            units,
            dropout=.4,
            return_state=True,
            return_sequences=True,
            name='encoder_lstm',
            kernel_initializer=tf.keras.initializers.GlorotNormal()
        )

    def call(self,encoder_inputs):
        self.inputs=encoder_inputs
        x=self.embedding(encoder_inputs)
        x=self.normalize(x)
        x=Dropout(.4)(x)
        encoder_outputs,encoder_state_h,encoder_state_c=self.lstm(x)
        self.outputs=[encoder_state_h,encoder_state_c]
        return encoder_state_h,encoder_state_c

encoder=Encoder(lstm_cells,embedding_dim,vocab_size,name='encoder')
encoder.call( [0])
```

Out[27]:
```
(<tf.Tensor: shape=(149, 256), dtype=float32, numpy=
 array([[ 0.00729879,  0.00261236,  0.27571142, ...,  0.18465781,
         -0.08156478, -0.11713018],
        [ 0.19469805,  0.05909612, -0.13187952, ..., -0.02056826,
         -0.00086583, -0.09591528],
        [ 0.17122817,  0.11873189, -0.2417493 , ...,  0.30304292,
         -0.06343494,  0.06781824],
        ...,
        [-0.09870663, -0.08098442, -0.03398362, ...,  0.09579566,
         -0.06533212, -0.20227465],
        [-0.03866765, -0.26193592, -0.01419795, ...,  0.03315157,
         -0.14567478, -0.0696206 ],
        [-0.05141401,  0.06219782, -0.04914484, ...,  0.06238681,
          0.0074725 , -0.00244139]], dtype=float32)>,
 <tf.Tensor: shape=(149, 256), dtype=float32, numpy=
 array([[ 0.01446833,  0.0064121 ,  0.4608011 , ...,  0.4364555 ,
         -0.13479029, -0.20563553],
        [ 0.3856144 ,  0.14027089, -0.21183857, ..., -0.04505116,
         -0.00143811, -0.15917444],
        [ 0.34029222,  0.2940804 , -0.38618863, ...,  0.7745548 ,
         -0.10513303,  0.11747956],
        ...,
        [-0.19619018, -0.17023808, -0.07251817, ...,  0.3087586 ,
         -0.13395691, -0.31459326],
        [-0.07272727, -0.58581984, -0.03437597, ...,  0.11165138,
         -0.29175073, -0.10364441],
        [-0.0987102 ,  0.13158694, -0.10850348, ...,  0.20872429,
          0.0154845 , -0.00364149]], dtype=float32)>)
```

Build Encoder## Build Decoder

In [28]:
```
class Decoder(tf.keras.models.Model):
    def __init__(self,units,embedding_dim,vocab_size,*args,**kwargs) -> Non
        super().__init__(*args,**kwargs)
        self.units=units
        self.embedding_dim=embedding_dim
        self.vocab_size=vocab_size
        self.embedding=Embedding(
            vocab_size,
            embedding_dim,
            name='decoder_embedding',
            mask_zero=True,
            embeddings_initializer=tf.keras.initializers.HeNormal()
        )
        self.normalize=LayerNormalization()
        self.lstm=LSTM(
            units,
            dropout=.4,
            return_state=True,
            return_sequences=True,
            name='decoder_lstm',
            kernel_initializer=tf.keras.initializers.HeNormal()
        )
        self.fc=Dense(
            vocab_size,
            activation='softmax',
            name='decoder_dense',
            kernel_initializer=tf.keras.initializers.HeNormal()
        )

    def call(self,decoder_inputs,encoder_states):
        x=self.embedding(decoder_inputs)
        x=self.normalize(x)
        x=Dropout(.4)(x)
        x,decoder_state_h,decoder_state_c=self.lstm(x,initial_state=encoder
        x=self.normalize(x)
        x=Dropout(.4)(x)
        return self.fc(x)

decoder=Decoder(lstm_cells,embedding_dim,vocab_size,name='decoder')
decoder( [1][:1] encoder( [0][:1]))
```

Out[28]:
```
<tf.Tensor: shape=(1, 30, 2443), dtype=float32, numpy=
array([[[2.9999198e-04, 1.6206181e-04, 9.0658228e-05, ...,
         7.7327830e-05, 8.0671057e-04, 2.4468001e-04],
        [3.1933453e-04, 1.4735044e-04, 3.2810498e-05, ...,
         4.4368464e-04, 2.4887559e-04, 1.3005025e-04],
        [8.7780919e-04, 1.9366412e-04, 1.9084984e-04, ...,
         3.4939087e-05, 9.4471485e-05, 1.9536835e-04],
        ...,
        [1.4621945e-04, 4.2559856e-04, 1.0154680e-03, ...,
         5.0618495e-05, 7.6976292e-05, 1.2466122e-04],
        [1.4621943e-04, 4.2559850e-04, 1.0154690e-03, ...,
         5.0618517e-05, 7.6976321e-05, 1.2466128e-04],
        [1.4621943e-04, 4.2559850e-04, 1.0154690e-03, ...,
         5.0618517e-05, 7.6976314e-05, 1.2466127e-04]]], dtype=float32)>
```

Build Training Model

In [29]:
```
class ChatBotTrainer(tf.keras.models.Model):
    def __init__(self,encoder,decoder,*args,**kwargs):
        super().__init__(*args,**kwargs)
        self.encoder=encoder
        self.decoder=decoder

    def loss_fn(self,y_true,y_pred):
        loss=self.loss(y_true,y_pred)
        mask=tf.math.logical_not(tf.math.equal(y_true,0))
        mask=tf.cast(mask,dtype=loss.dtype)
        loss*=mask
        return tf.reduce_mean(loss)

    def accuracy_fn(self,y_true,y_pred):
        pred_values = tf.cast(tf.argmax(y_pred, axis=-1), dtype='int64')
        correct = tf.cast(tf.equal(y_true, pred_values), dtype='float64')
        mask = tf.cast(tf.greater(y_true, 0), dtype='float64')
        n_correct = tf.keras.backend.sum(mask * correct)
        n_total = tf.keras.backend.sum(mask)
        return n_correct / n_total

    def call(self,inputs):
        encoder_inputs,decoder_inputs=inputs
        encoder_states=self.encoder(encoder_inputs)
        return self.decoder(decoder_inputs,encoder_states)

    def train_step(self,batch):
        encoder_inputs,decoder_inputs,y=batch
        with tf.GradientTape() as tape:
            encoder_states=self.encoder(encoder_inputs,training=True)
            y_pred=self.decoder(decoder_inputs,encoder_states,training=True
            loss=self.loss_fn(y,y_pred)
            acc=self.accuracy_fn(y,y_pred)

        variables=self.encoder.trainable_variables+self.decoder.trainable_v
        grads=tape.gradient(loss,variables)
        self.optimizer.apply_gradients(zip(grads,variables))
        metrics={'loss':loss,'accuracy':acc}
        return metrics

    def test_step(self,batch):
        encoder_inputs,decoder_inputs,y=batch
        encoder_states=self.encoder(encoder_inputs,training=True)
        y_pred=self.decoder(decoder_inputs,encoder_states,training=True)
        loss=self.loss_fn(y,y_pred)
        acc=self.accuracy_fn(y,y_pred)
        metrics={'loss':loss,'accuracy':acc}
        return metrics
```

In [30]:
```
model=ChatBotTrainer(encoder,decoder,name='chatbot_trainer')
model.compile(
    loss=tf.keras.losses.SparseCategoricalCrossentropy(),
    optimizer=tf.keras.optimizers.Adam(learning_rate=learning_rate),
    weighted_metrics=['loss','accuracy']
)
model(_[:2])
```

Out[30]:
```
<tf.Tensor: shape=(149, 30, 2443), dtype=float32, numpy=
array([[[2.99992273e-04, 1.62061726e-04, 9.06581845e-05, ...,
         7.73278225e-05, 8.06710916e-04, 2.44680123e-04],
        [3.19334533e-04, 1.47350293e-04, 3.28105125e-05, ...,
         4.43684548e-04, 2.48875498e-04, 1.30050190e-04],
        [8.77808721e-04, 1.93663916e-04, 1.90849911e-04, ...,
         3.49390684e-05, 9.44714266e-05, 1.95368324e-04],
        ...,
        [1.46219434e-04, 4.25598118e-04, 1.01546827e-03, ...,
         5.06184915e-05, 7.69762482e-05, 1.24661266e-04],
        [1.46219434e-04, 4.25598118e-04, 1.01546827e-03, ...,
         5.06184915e-05, 7.69762482e-05, 1.24661266e-04],
        [1.46219434e-04, 4.25598118e-04, 1.01546827e-03, ...,
         5.06184915e-05, 7.69762482e-05, 1.24661266e-04]],

       [[7.65215838e-04, 4.97353540e-05, 1.43004276e-04, ...,
         5.23312992e-05, 9.79369273e-04, 1.76479819e-03],
        [1.06070936e-03, 8.07600678e-04, 2.43992108e-04, ...,
         4.45704973e-05, 2.61238136e-04, 1.91518280e-03],
        [3.21416272e-04, 2.11614347e-03, 2.29154262e-04, ...,
         1.62517405e-04, 1.50816137e-04, 5.45158749e-04],
        ...,
        [1.06548439e-04, 3.48956644e-04, 8.72848905e-04, ...,
         6.20231294e-05, 2.38176508e-04, 5.16010077e-05],
        [1.06548439e-04, 3.48956644e-04, 8.72848905e-04, ...,
         6.20231294e-05, 2.38176508e-04, 5.16010077e-05],
        [1.06548439e-04, 3.48956644e-04, 8.72848905e-04, ...,
         6.20231294e-05, 2.38176508e-04, 5.16010077e-05]],

       [[1.15758250e-03, 5.19195091e-05, 3.05715541e-04, ...,
         8.22967195e-05, 1.17772934e-03, 1.12907856e-03],
        [1.36418070e-03, 3.11160955e-04, 1.81025884e-04, ...,
         9.52949398e-04, 4.04806458e-04, 8.48339172e-04],
        [3.77289864e-04, 2.73013744e-03, 2.60455854e-04, ...,
         2.49268109e-04, 4.70473227e-04, 4.37766605e-04],
        ...,
        [2.38431330e-05, 1.47987448e-04, 7.52890832e-04, ...,
         1.51785658e-04, 6.52307572e-05, 4.44584293e-05],
        [2.38431330e-05, 1.47987448e-04, 7.52890832e-04, ...,
         1.51785658e-04, 6.52307644e-05, 4.44584293e-05],
        [2.38431330e-05, 1.47987448e-04, 7.52890832e-04, ...,
         1.51785658e-04, 6.52307572e-05, 4.44584293e-05]],

       ...,

       [[7.93311774e-05, 5.77299070e-05, 1.21629062e-04, ...,
         3.27706875e-05, 2.71701632e-04, 1.12613174e-03],
        [1.06883133e-04, 7.36106245e-04, 1.79120223e-04, ...,
         1.62239467e-05, 2.06230718e-04, 2.38520021e-04],
        [1.68044397e-04, 9.27104411e-05, 3.51270028e-05, ...,
         4.82014802e-05, 9.99158510e-05, 4.26230981e-04],
        ...,
        [2.30115096e-04, 5.93274599e-04, 6.09832990e-04, ...,
         3.74064039e-05, 1.62934622e-04, 5.49282449e-05],
        [2.30115096e-04, 5.93274599e-04, 6.09832990e-04, ...,
         3.74064039e-05, 1.62934622e-04, 5.49282449e-05],
        [2.30115096e-04, 5.93274599e-04, 6.09832990e-04, ...,
         3.74064039e-05, 1.62934622e-04, 5.49282449e-05]],

       [[2.18140267e-04, 4.28628467e-04, 1.31877867e-04, ...,
         1.62192973e-05, 2.22797564e-04, 1.35396095e-03],
```

```
          [2.14760005e-03, 7.12900481e-04, 3.23126951e-05, ...,
           1.28536794e-05, 2.18208908e-04, 3.09290452e-04],
          [3.59534926e-04, 9.73579183e-04, 9.20535895e-05, ...,
           4.48041828e-05, 4.79357113e-04, 2.76508625e-04],
          ...,
          [2.38944747e-04, 1.27456791e-03, 3.66528373e-04, ...,
           1.55934933e-04, 3.66495253e-04, 8.46676339e-05],
          [2.38944747e-04, 1.27456791e-03, 3.66528373e-04, ...,
           1.55934933e-04, 3.66495253e-04, 8.46676339e-05],
          [2.38944747e-04, 1.27456791e-03, 3.66528373e-04, ...,
           1.55934933e-04, 3.66495253e-04, 8.46676339e-05]],

         [[1.00405006e-04, 4.22082143e-04, 1.27401087e-04, ...,
           2.40375593e-05, 1.92060194e-04, 1.49707031e-03],
          [1.34221662e-03, 1.74594126e-04, 5.81798435e-04, ...,
           1.85912977e-05, 6.43423991e-04, 7.43874290e-04],
          [1.03806658e-03, 4.34423127e-04, 6.34534590e-05, ...,
           1.13429036e-04, 1.16517383e-03, 5.70863951e-04],
          ...,
          [1.34756745e-04, 5.30583609e-04, 6.21304964e-04, ...,
           6.84591432e-05, 8.15877283e-05, 1.14165341e-04],
          [1.34756745e-04, 5.30583842e-04, 6.21304964e-04, ...,
           6.84591723e-05, 8.15878011e-05, 1.14165450e-04],
          [1.34756745e-04, 5.30583842e-04, 6.21304964e-04, ...,
           6.84591723e-05, 8.15878011e-05, 1.14165450e-04]]], dtype=float32)
>
```

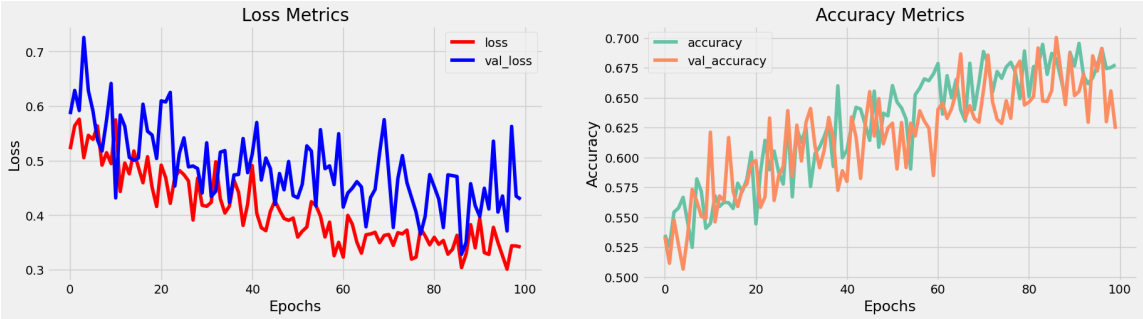
Train Model

In [34]:
```
history=model.fit(
    train_data,
    epochs=100,
    validation_data=val_data,
    callbacks=[
        tf.keras.callbacks.TensorBoard(log_dir='logs'),
        tf.keras.callbacks.ModelCheckpoint('ckpt',verbose=1,save_best_only=
    ]
)
```

```
Epoch 1/100
23/23 [==============================] - ETA: 0s - loss: 0.5471 - accur
acy: 0.5353
Epoch 1: val_loss improved from inf to 0.58522, saving model to ckpt
WARNING:tensorflow:Model's `__init__()` arguments contain non-serializa
ble objects. Please implement a `get_config()` method in the subclassed
Model for proper saving and loading. Defaulting to empty config.

WARNING:tensorflow:Model's `__init__()` arguments contain non-serializa
ble objects. Please implement a `get_config()` method in the subclassed
Model for proper saving and loading. Defaulting to empty config.

WARNING:tensorflow:Model's `__init__()` arguments contain non-serializa
ble objects. Please implement a `get_config()` method in the subclassed
Model for proper saving and loading. Defaulting to empty config.

WARNING:tensorflow:Model's `__init__()` arguments contain non-serializa
ble objects. Please implement a `get_config()` method in the subclassed
Model for proper saving and loading. Defaulting to empty config.
```

Visualize Metrics

In [35]:
```
fig,ax=plt.subplots(nrows=1,ncols=2,figsize=(20,5))
ax[0].plot(history.history['loss'],label='loss',c='red')
ax[0].plot(history.history['val_loss'],label='val_loss',c = 'blue')
ax[0].set_xlabel('Epochs')
ax[1].set_xlabel('Epochs')
ax[0].set_ylabel('Loss')
ax[1].set_ylabel('Accuracy')
ax[0].set_title('Loss Metrics')
ax[1].set_title('Accuracy Metrics')
ax[1].plot(history.history['accuracy'],label='accuracy')
ax[1].plot(history.history['val_accuracy'],label='val_accuracy')
ax[0].legend()
ax[1].legend()
plt.show()
```



Save Model

In [38]:
```
model.load_weights('ckpt')
model.save('models',save_format='tf')
```

```
WARNING:tensorflow:Model's `__init__()` arguments contain non-serializable objects. Please implement a `get_config()` method in the subclassed Model for proper saving and loading. Defaulting to empty config.

WARNING:tensorflow:Model's `__init__()` arguments contain non-serializable objects. Please implement a `get_config()` method in the subclassed Model for proper saving and loading. Defaulting to empty config.

WARNING:tensorflow:Model's `__init__()` arguments contain non-serializable objects. Please implement a `get_config()` method in the subclassed Model for proper saving and loading. Defaulting to empty config.

WARNING:tensorflow:Model's `__init__()` arguments contain non-serializable objects. Please implement a `get_config()` method in the subclassed Model for proper saving and loading. Defaulting to empty config.

INFO:tensorflow:Assets written to: models\assets

INFO:tensorflow:Assets written to: models\assets

WARNING:tensorflow:Model's `__init__()` arguments contain non-serializable objects. Please implement a `get_config()` method in the subclassed Model for proper saving and loading. Defaulting to empty config.

WARNING:tensorflow:Model's `__init__()` arguments contain non-serializable objects. Please implement a `get_config()` method in the subclassed Model for proper saving and loading. Defaulting to empty config.

WARNING:tensorflow:Model's `__init__()` arguments contain non-serializable objects. Please implement a `get_config()` method in the subclassed Model for proper saving and loading. Defaulting to empty config.

WARNING:tensorflow:Model's `__init__()` arguments contain non-serializable objects. Please implement a `get_config()` method in the subclassed Model for proper saving and loading. Defaulting to empty config.
```

In [39]:
```
for idx,i in enumerate(model.layers):
    print('Encoder layers:' if idx==0 else 'Decoder layers: ')
    for j in i.layers:
        print(j)
    print('---------------------')
```

```
Encoder layers:
<keras.src.layers.core.embedding.Embedding object at 0x000002AAA5244AD0>
<keras.src.layers.normalization.layer_normalization.LayerNormalization object at 0x000002AAA5770750>
<keras.src.layers.rnn.lstm.LSTM object at 0x000002AAA1D62C50>
---------------------
Decoder layers:
<keras.src.layers.core.embedding.Embedding object at 0x000002AAA1D56450>
<keras.src.layers.normalization.layer_normalization.LayerNormalization object at 0x000002AAA1D03F10>
<keras.src.layers.rnn.lstm.LSTM object at 0x000002AAA1D61310>
<keras.src.layers.core.dense.Dense object at 0x000002AAA1CCDA50>
---------------------
```

Create Inference Model

In [40]:
```
class ChatBot(tf.keras.models.Model):
    def __init__(self,base_encoder,base_decoder,*args,**kwargs):
        super().__init__(*args,**kwargs)
        self.encoder,self.decoder=self.build_inference_model(base_encoder,b

    def build_inference_model(self,base_encoder,base_decoder):
        encoder_inputs=tf.keras.Input(shape=(None,))
        x=base_encoder.layers[0](encoder_inputs)
        x=base_encoder.layers[1](x)
        x,encoder_state_h,encoder_state_c=base_encoder.layers[2](x)
        encoder=tf.keras.models.Model(inputs=encoder_inputs,outputs=[encode

        decoder_input_state_h=tf.keras.Input(shape=(lstm_cells,))
        decoder_input_state_c=tf.keras.Input(shape=(lstm_cells,))
        decoder_inputs=tf.keras.Input(shape=(None,))
        x=base_decoder.layers[0](decoder_inputs)
        x=base_encoder.layers[1](x)
        x,decoder_state_h,decoder_state_c=base_decoder.layers[2](x,initial_
        decoder_outputs=base_decoder.layers[-1](x)
        decoder=tf.keras.models.Model(
            inputs=[decoder_inputs,[decoder_input_state_h,decoder_input_sta
            outputs=[decoder_outputs,[decoder_state_h,decoder_state_c]],nam
        )
        return encoder,decoder

    def summary(self):
        self.encoder.summary()
        self.decoder.summary()

    def softmax(self,z):
        return np.exp(z)/sum(np.exp(z))

    def sample(self,conditional_probability,temperature=0.5):
        conditional_probability = np.asarray(conditional_probability).astyp
        conditional_probability = np.log(conditional_probability) / tempera
        reweighted_conditional_probability = self.softmax(conditional_proba
        probas = np.random.multinomial(1, reweighted_conditional_probabilit
        return np.argmax(probas)

    def preprocess(self,text):
        text=clean_text(text)
        seq=np.zeros((1,max_sequence_length),dtype=np.int32)
        for i,word in enumerate(text.split()):
            seq[:,i]=sequences2ids(word).numpy()[0]
        return seq

    def postprocess(self,text):
        text=re.sub(' - ','-',text.lower())
        text=re.sub(' [.] ','. ',text)
        text=re.sub(' [1] ','1',text)
        text=re.sub(' [2] ','2',text)
        text=re.sub(' [3] ','3',text)
        text=re.sub(' [4] ','4',text)
        text=re.sub(' [5] ','5',text)
        text=re.sub(' [6] ','6',text)
        text=re.sub(' [7] ','7',text)
        text=re.sub(' [8] ','8',text)
        text=re.sub(' [9] ','9',text)
        text=re.sub(' [0] ','0',text)
        text=re.sub(' [,] ',', ',text)
        text=re.sub(' [?] ','? ',text)
```

```
        text=re.sub(' [!] ','! ',text)
        text=re.sub(' [$] ','$ ',text)
        text=re.sub(' [&] ','& ',text)
        text=re.sub(' [/] ','/ ',text)
        text=re.sub(' [:] ',': ',text)
        text=re.sub(' [;] ','; ',text)
        text=re.sub(' [*] ','* ',text)
        text=re.sub(' [\'] ','\'',text)
        text=re.sub(' [\"] ','\"',text)
        return text

    def call(self,text,config=None):
        input_seq=self.preprocess(text)
        states=self.encoder(input_seq,training=False)
        target_seq=np.zeros((1,1))
        target_seq[:,:]=sequences2ids(['<start>']).numpy()[0][0]
        stop_condition=False
        decoded=[]
        while not stop_condition:
            decoder_outputs,new_states=self.decoder([target_seq,states],tra
#             index=tf.argmax(decoder_outputs[:,-1,:],axis=-1).numpy().item
            index=self.sample(decoder_outputs[0,0,:]).item()
            word=ids2sequences([index])
            if word=='<end> ' or len(decoded)>=max_sequence_length:
                stop_condition=True
            else:
                decoded.append(index)
                target_seq=np.zeros((1,1))
                target_seq[:,:]=index
                states=new_states
        return self.postprocess(ids2sequences(decoded))

chatbot=ChatBot(model.encoder,model.decoder,name='chatbot')
chatbot.summary()
```

```
Model: "chatbot_encoder"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, None)]            0

 encoder_embedding (Embeddi  (None, None, 256)         625408
 ng)

 layer_normalization_2 (Lay  (None, None, 256)         512
 erNormalization)

 encoder_lstm (LSTM)         [(None, None, 256),       525312
                              (None, 256),
                              (None, 256)]

=================================================================
Total params: 1151232 (4.39 MB)
Trainable params: 1151232 (4.39 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
Model: "chatbot_decoder"
__________________________________________________________________________________________________
 Layer (type)                Output Shape                 Param #   Connec
ted to
==================================================================================================
 input_4 (InputLayer)        [(None, None)]               0         []

 decoder_embedding (Embeddi  (None, None, 256)            625408    ['inpu
t_4[0][0]']
 ng)

 layer_normalization_2 (Lay  (None, None, 256)            512       ['deco
der_embedding[0][0]']
 erNormalization)

 input_2 (InputLayer)        [(None, 256)]                0         []

 input_3 (InputLayer)        [(None, 256)]                0         []

 decoder_lstm (LSTM)         [(None, None, 256),          525312    ['laye
r_normalization_2[1][0]'
                              (None, 256),                           , 'inp
ut_2[0][0]',
                              (None, 256)]                            'inpu
t_3[0][0]']

 decoder_dense (Dense)       (None, None, 2443)           627851    ['deco
der_lstm[0][0]']

==================================================================================================
Total params: 1779083 (6.79 MB)
Trainable params: 1779083 (6.79 MB)
Non-trainable params: 0 (0.00 Byte)
__________________________________________________________________________________________________
```

In [41]:
```
tf.keras.utils.plot_model(chatbot.encoder,to_file='encoder.png',show_shapes
```

```
You must install pydot (`pip install pydot`) and install graphviz (see ins
tructions at https://graphviz.gitlab.io/download/) (https://graphviz.gitla
b.io/download/)) for plot_model to work.
```

In [ ]:
```
tf.keras.utils.plot_model(chatbot.decoder,to_file='decoder.png',show_shapes
```

Time to Chat

In [42]:
```
def print_conversation(texts):
    for text in texts:
        print(f'You: {text}')
        print(f'Bot: {chatbot(text)}')
        print('========================')
```

In [43]:
```
print_conversation([
    'hi',
    'do yo know me?',
    'what is your name?',
    'you are bot?',
    'hi, how are you doing?',
    "i'm pretty good. thanks for asking.",
    "Don't ever be in a hurry",
    '''I'm gonna put some dirt in your eye ''',
    '''You're trash ''',
    '''I've read all your research on nano-technology ''',
    '''You want forgiveness? Get religion''',
    '''While you're using the bathroom, i'll order some food.''',
    '''Wow! that's terrible.''',
    '''We'll be here forever.''',
    '''I need something that's reliable.''',
    '''A speeding car ran a red light, killing the girl.''',
    '''Tomorrow we'll have rice and fish for lunch.''',
    '''I like this restaurant because they give you free bread.'''
])
```

```
Bot: don't order for me. I'll have a ticket in my life.
========================
You: Wow! that's terrible.
Bot: never's a good deal.
========================
You: We'll be here forever.
Bot: we'll be here forever.
========================
You: I need something that's reliable.
Bot: you're not going to be buried.
========================
You: A speeding car ran a red light, killing the girl.
Bot: what happened?
========================
You: Tomorrow we'll have rice and fish for lunch.
Bot: i'll get a bucket.
========================
You: I like this restaurant because they give you free bread.
Bot: well, i do. it's so good for the people.
========================
```