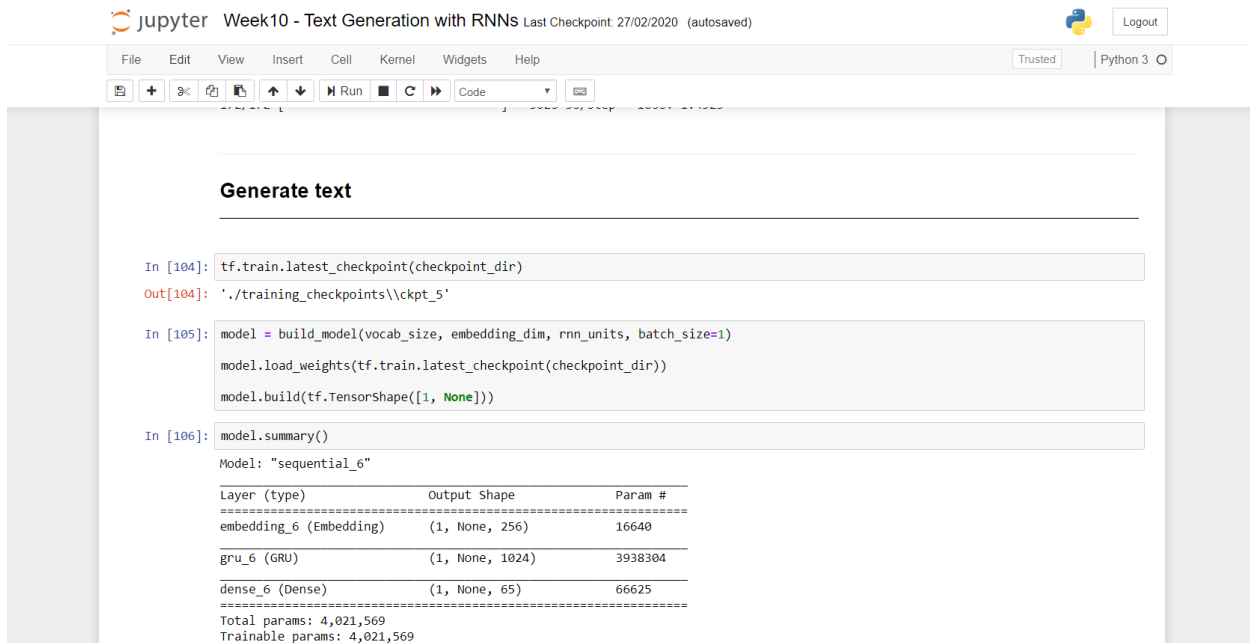


Text generation Prototype

In this section of the report, I have successfully trained my model and I have performed my first text generation operation.

Step 1: Load the model training checkpoint and model weights



The screenshot shows a Jupyter Notebook titled "Week10 - Text Generation with RNNs". The interface includes a menu bar (File, Edit, View, Insert, Cell, Kernel, Widgets, Help) and a toolbar with icons for file operations, running cells, and code execution. The notebook content is as follows:

```
In [104]: tf.train.latest_checkpoint(checkpoint_dir)
Out[104]: './training_checkpoints\\ckpt_5'
```

```
In [105]: model = build_model(vocab_size, embedding_dim, rnn_units, batch_size=1)
          model.load_weights(tf.train.latest_checkpoint(checkpoint_dir))
          model.build(tf.TensorShape([1, None]))
```

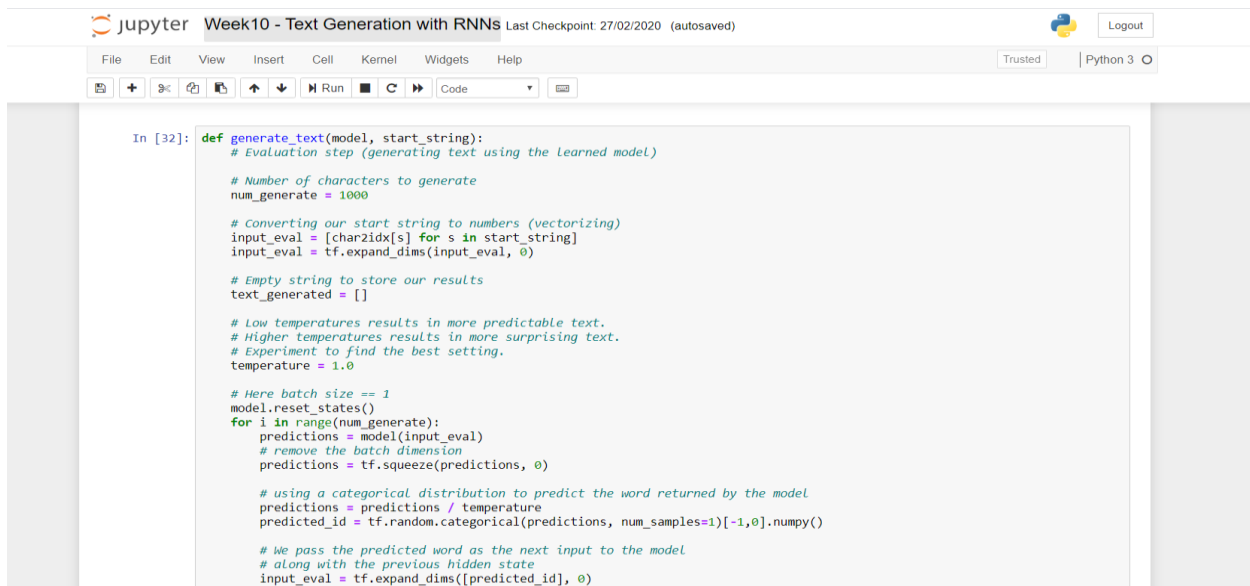
```
In [106]: model.summary()
```

Model: "sequential_6"

Layer (type)	Output Shape	Param #
embedding_6 (Embedding)	(1, None, 256)	16640
gru_6 (GRU)	(1, None, 1024)	3938304
dense_6 (Dense)	(1, None, 65)	66625

Total params: 4,021,569
Trainable params: 4,021,569

Step 2: Define the text generation function



The screenshot shows the same Jupyter Notebook interface with the following code cell:

```
In [32]: def generate_text(model, start_string):
          # Evaluation step (generating text using the Learned model)

          # Number of characters to generate
          num_generate = 1000

          # Converting our start string to numbers (vectorizing)
          input_eval = [char2idx[s] for s in start_string]
          input_eval = tf.expand_dims(input_eval, 0)

          # Empty string to store our results
          text_generated = []

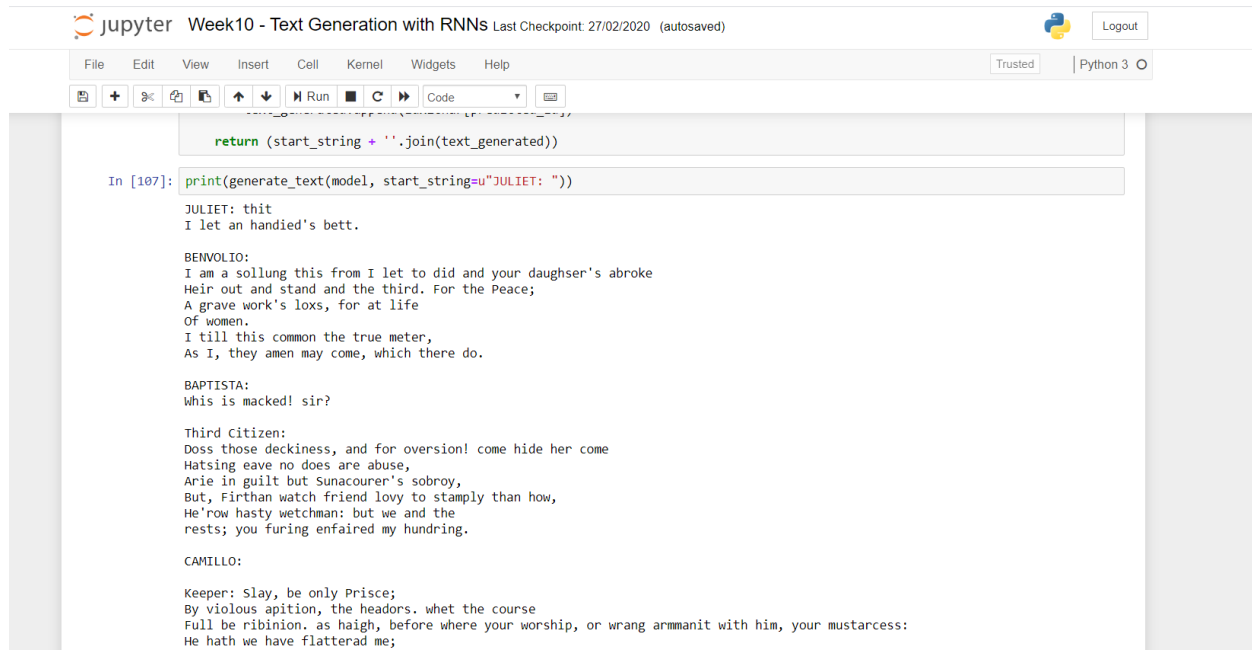
          # Low temperatures results in more predictable text.
          # Higher temperatures results in more surprising text.
          # Experiment to find the best setting.
          temperature = 1.0

          # Here batch size == 1
          model.reset_states()
          for i in range(num_generate):
              predictions = model(input_eval)
              # remove the batch dimension
              predictions = tf.squeeze(predictions, 0)

              # using a categorical distribution to predict the word returned by the model
              predictions = predictions / temperature
              predicted_id = tf.random.categorical(predictions, num_samples=1)[-1,0].numpy()

              # We pass the predicted word as the next input to the model
              # along with the previous hidden state
              input_eval = tf.expand_dims([predicted_id], 0)
```

Step 3: Run the text generator with a token word “JULIET”



```
return (start_string + ''.join(text_generated))
```

```
In [107]: print(generate_text(model, start_string=u"JULIET: "))
```

JULIET: thit
I let an handied's bett.

BENVOLIO:
I am a sollung this from I let to did and your daughser's abroke
Heir out and stand and the third. For the Peace;
A grave work's loxs, for at life
Of women.
I till this common the true meter,
As I, they amen may come, which there do.

BAPTISTA:
Whis is macked! sir?

Third Citizen:
Doss those deckiness, and for overion! come hide her come
Hatsing eave no does are abuse,
Arie in guilt but Sunacourer's sobroy,
But, Firthan watch friend lovy to stamply than how,
He'row hasty wetchman: but we and the
rests; you furing enfaired my hundring.

CAMILLO:
Keeper: Slay, be only Prisce;
By violous apition, the headors. whet the course
Full be ribinion. as haigh, before where your worship, or wrang armanit with him, your mustarcess:
He hath we have flattered me;

I have been able to successfully generate sentences based on our initial dataset input.

Furthermore some other concepts such as LSTM layers can be introduced into this type of RNN model, although it would require more work, a different preprocessing technique, and addition of layers to the model architecture, this would be a possibly good improvement to the model.