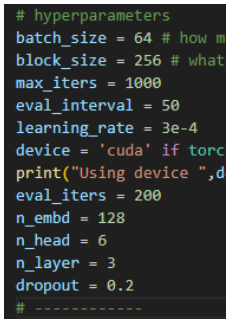
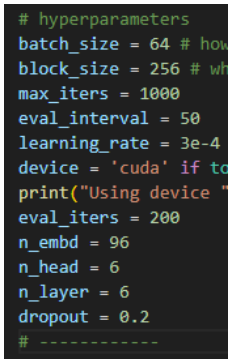
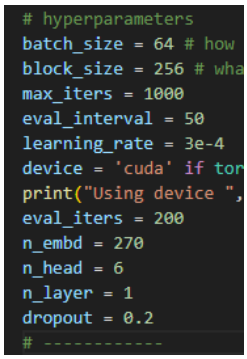
i. Part i
   a. The first dataset “input_childSpeech_trainingSet.txt” contains 10,000 lines of phrases one would likely hear from a child, with on average about 20 characters / 5 words per entry, most entries are repeated multiple times throughout the list
   The second dataset “input_child_speech_testSet.txt” includes a very similar dataset to the first one, but is only 1,000 lines. While it still contains repeats, it contains much less than the first dataset
   The third dataset “input_shakespeare.txt” contains 40,000 lines of shakespeare, which doesn’t seem to repeat any lines(unless they naturally would in the writing), the play does include line breaks and names for each new line read by the performers. The lines are on average about 40 characters long and make up about 7 words each.



   b. For downsizing the model I reduced the following:
   max_iters was reduced to 1000 to save me time
   n_embd and n_layer were both reduced in order to reduce the amount of parameters
   n_embd was reduced from 384 -> 128, this reduced the number of parameters to 1/9 due to the O(N^2) complexity it contains
   n_layer was reduced from 6 -> 3, this reduced the number of parameters to ½
   no other variables directly change the number of parameters, while they change other things
   c. For my 2 other tests I reduced the n_embd only and the n_layer only respectively. While still remaining below 1M parameters

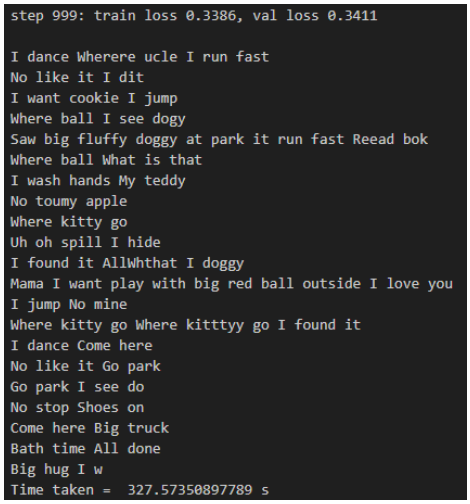


n_embd reduced, for this test I reduced n_embd to from 384->96, which reduced the number of parameters to 1/16th of the original(0.7m parameters)

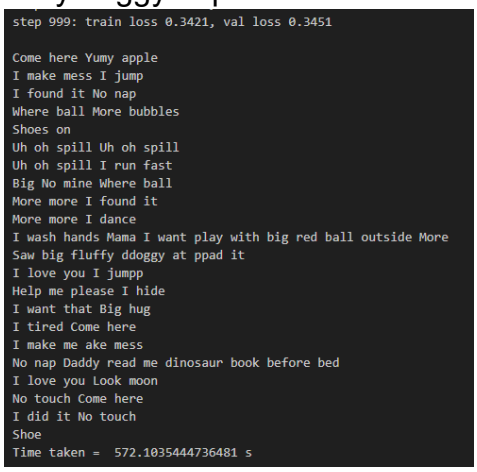


n_layer reduced, for this test I reduced n_layer from 6 -> 1, which reduced the number of parameters to 1.9m, while this wasn’t below the 1m

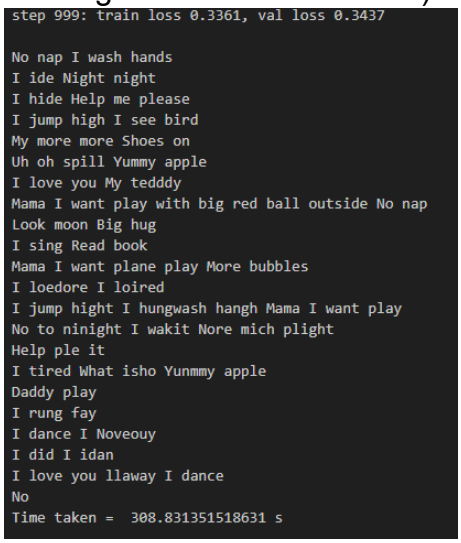
objective, it couldn't be reduced any further through this change, thus it had to be accepted as is for this test.



The first downsizing had a train loss of 0.3386 and a val loss of 0.3411, being a difference of 0.0025. It wrote good phrases, with only occasional spelling mistakes, but most of the sentences made sense e.g.(Saw big fluffy doggy at park it run fast Reead bok)



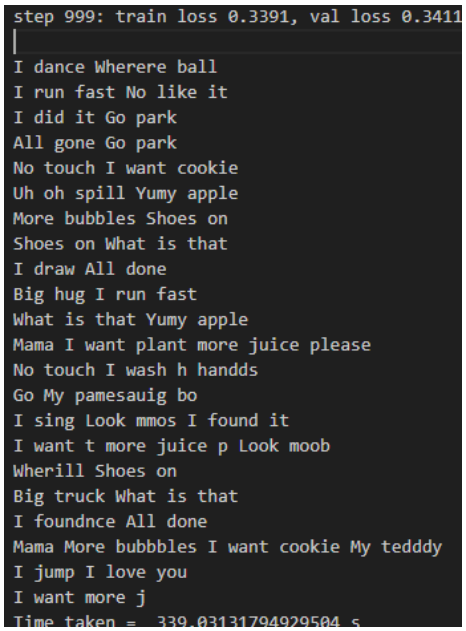
The second downsizing had a train loss of 0.3421 and a val loss of 0.3451 being a difference of 0.003,having the largest of both loss this might suggest underfitting. It wrote near-perfect sentences with very very few mistakes in spelling e.g.(Come here Yumy apple, I wash hands Mama I want play with big red ball outside More)



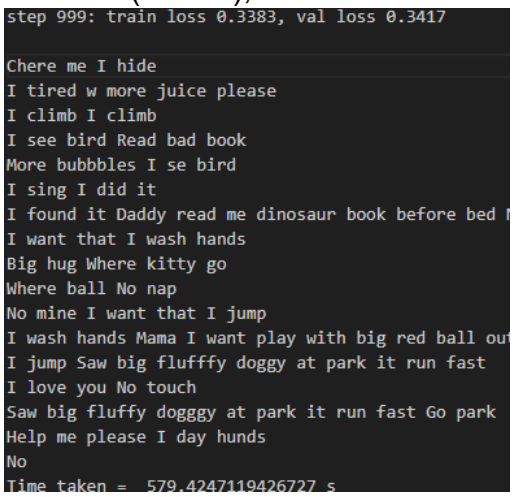
The third downsizing had a train loss of 0.3361 and a val loss of 0.3437, being a difference of 0.0076 , by far the largest difference which might suggest overfitting. It wrote multiple lines of pure gibberish, with few real sentences, e.g.(No to ninight I wakit Nore mich plight), this shows that it may have

suffered from overfitting, which is very likely from its n_layer being reduced to 1, therefore it couldnt make find of the complex patterns.

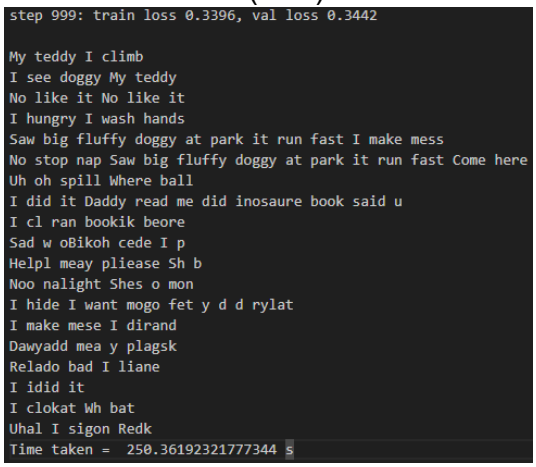
d. In order to introduce a bias I modified the nn.linear calls to include bias=True,



The first model increased its train loss from 0.3386 -> 0.3391(+0.1%), but achieved an identical val loss of 0.3411(0%).



The second model lowered its train loss from 0.3421 -> 0.3383(-1%) and lowered its val loss from 0. 3451-> 0.3417(-1%)



The third model increased its train loss from 0.3361 -> 0.3396(+1%) but increased its val loss from 0.3437 -> 0.3442(0.1%) While the differences of the individual loss werent substantial, the gaps between them were majorly affected

The first model reduced its difference from 0.0025 -> 0.002, a reduction of 20%

The second model increasedits difference from 0.0030 -> 0.0034, an increase of 13%
The third model reduced its difference from 0.0076 -> 0.0046, a reduction of 40%
The third model had the biggest change in difference, suggesting that it may have become less overfit but many of its sentences remained a gibberish mess.

All 3 models created similar quality of sentences than they had before.
Adding bias allowed the parameters to shift the attentions appropriately, this helped the models with learning, especially with the risk of overfitting.

e.

```
def forward(self, x):
    return x
    x = x + self.sa(self.ln1(x))
    x = x + self.ffwd(self.ln2(x))
```

```
step 999: train loss 2.0121, val loss 2.0136

Cook juce de d cllestig
Whtosimadreareselle
I nt headigrere p hufokideas plefalapane bas inest hume daumoshut p are re ngy drp tsed llloe rp
Mo pliruchtigy s here s sas I Lok tire
Wh Whomande bise
I p ie pplire ble Alp was s pleall Norumesty a fy Mak wangg ay k Alo de the
I berumitee
Goreano bides mpatry me
Wh n y lle
Big himeffarerawawalea Bine bas I Nore
Shuch I se
I juie l Whithau fo flaithe I k
Uhaty I ogyomohubide paple I bind bb Whaiteatitrpllllilllad hanan
Day ckitirere Coedy
I te
Nove dd
Time taken =  94.41125679016113 s
```

Removing the skip connections functionality from the code causes monumental increases to train loss and val loss(up to about 2) and causes completely incoherent outputs, while the model does train much faster, it isnt worth the loss in quality.
The main advantage of the skip connection is to let the self-attention mechanism focus the relationships between words without having to preserve the word and position encoding.

ii. Testing the model.

```
torch.save(model.state_dict(), "model.pth")
```

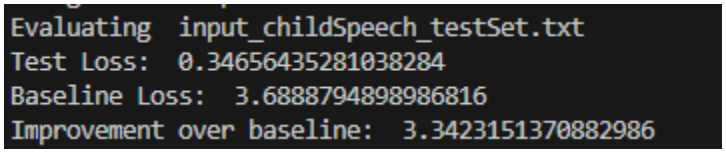
```
model.load_state_dict(torch.load('model.pth'))
```

These pieces of code allowed me to save a models state and later load it, this helped to speed up testing as I didn't need to wait for the entire training process every time it was tested.

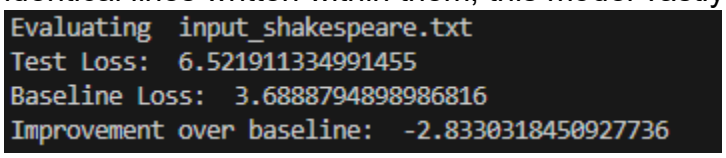
```
def evaluate_loss(model, data):
    model.eval()
    total_loss = 0
    num_batches = 0

    for k in range(eval_iters):
        ix = torch.randint(0, len(data) - block_size, (batch_size,))

        x = torch.stack([data[i:i+block_size] for i in ix]).to(device)
        y = torch.stack([data[i+1:i+block_size+1] for i in ix]).to(device)

        logits, loss = model(x, y)
        total_loss += loss.item()
        num_batches += 1

    model.train()
    return total_loss / num_batches
```

This code allowed me to evaluate the loss of the model against a set of data, by recycling the get_batch and estimate_loss functions, I was able to create a function that would take a model and encoded data and output an average loss value.

For this test I used the first trained model(n_embd=128, n_layer=3), in its biased form, as this had the best val loss and smallest difference in losses.

a. 

The test loss was slightly higher than the train loss and val loss that had previously been calculated. This is clearly due to the fact that the childSpeech_testSet and childSpeech_trainingSet had almost identical lines written within them, this model vastly outperformed the baseline test.

b. 

The test loss here was incredibly terrible, even significantly worse than the baseline test. This proves that the model is specialised in child speech, and cannot reliably replicate other speaking styles
The massive difference between the 2 test results clearly shows that this model is well trained in child speech and can consistently replicate it, but is worse than random at other speech patterns
This pipeline could be used for creating childrens stories, simplifying concepts down to childs speech etc. But the model can also be trained on other specific speech types, thus creating any required speech pattern as long as there is enough training data.