Apurva Mandalika
334003575

CSCE 636: Deep Learning (Spring 2024)

HW3 Report

---

1. We need to analyze the autoregressive language model and understand how the number of entries in the probability tables grows exponentially with the value of n.

The autoregressive language model, as given by equation (12.31) in the textbook "Deep Learning: Foundations and Concepts", can be represented as:

$P(x_n|x_1,x_2,...,x_{n-1})$

Where $x_1,x_2,...,x_{n-1}$ are the previous n-1 words in the sequence, and we want to predict the probability distribution over the next word x_n .

In this model, the conditional probability $P(x_n|x_1,x_2,...,x_{n-1})$ is represented by a probability table. Each entry in this table represents the probability of observing x_n given the sequence $x_1,x_2,...,x_{n-1}$.

Let's analyze the number of entries in these probability tables as n increases:

- **a.** For the first word (x_1), there are no previous words, so there is only one entry in the probability table, i.e., $P(x_1)$.
- **b.** For the second word (x_2), there is one previous word (x_1). Therefore, the probability table will have as many entries as there are possible values of x_1. If there are V possible values for each word in the vocabulary, then the probability table for x_2 will have V entries.
- **c.** For the third word (x_3), there are two previous words (x_1 and x_2). Therefore, the probability table will have as many entries as there are possible combinations of x_1 and x_2, which is V^2.
- **d.** Similarly, for the nth word (x_n), there are n-1 previous words. Therefore, the probability table will have as many entries as there are possible combinations of the previous n-1 words, which is V^{n-1}.

As we can see, the number of entries in the probability tables grows exponentially with the value of n, specifically as V^{n-1}, where V is the size of the vocabulary. This exponential growth occurs because each additional word in the sequence multiplies the number of possible combinations of previous words by the size of the vocabulary.

Therefore, it is evident that the number of entries in the probability tables increases rapidly as the value of n increases, making the model computationally expensive to train and use for large values of n. This exponential growth in the number of parameters is one of the challenges of scaling autoregressive language models.

2.

a. After the word embedding layer, the representations of words like "bank" are not inherently able to distinguish between the two occurrences in the given example sentences. This is because word embeddings typically represent words based on their context within a large corpus of text. In other words, words that appear in similar contexts are represented by similar vectors in the embedding space. In the example sentences provided, the word "bank" appears in similar contexts (as a location where one can get something), and thus, their embeddings might not capture the specific meaning or usage of "bank" in each sentence.
For instance, in both sentences, "bank" follows the preposition "from" or "to" and is associated with the action of getting something. Therefore, the word embeddings for "bank" in both sentences might end up being quite similar since they are influenced by the context of being a destination for an action rather than capturing the semantic differences between the two types of banks (the financial institution and the side of a river).

b. However, after applying a transformer layer on top of the word embeddings, there is a potential for the representations of "bank" to distinguish between the two occurrences in the example sentences. The transformer layer, through its self-attention mechanism, can capture more nuanced relationships between words in a sentence and encode contextual information more effectively. In the given example sentences, the transformer layer can learn to attend more to the words preceding "bank" to understand its specific usage.
For example, in the sentence "to get cash from the bank," the transformer layer might attend more to the word "cash" and learn that "bank" here refers to a financial institution where one can withdraw or deposit money. On the other hand, in the sentence "to get to the other bank," the transformer layer might attend more to the words "the other" and "across the river" and learn that "bank" here refers to the side of the river. By considering the surrounding context more effectively and capturing the dependencies between words, the transformer layer has the potential to produce more meaningful representations for words like "bank," allowing it to distinguish between different occurrences based on their semantic usage in context. Therefore, while the word embeddings alone might struggle to differentiate between the two occurrences of "bank," the transformer layer enhances the model's ability to do so by leveraging contextual information more effectively.

3. Coding Task

a. A tokenizer is used to break down a piece of text into smaller units, typically words or sub-words. It converts raw text data into a format that can be processed by machine learning models.
In the provided code, the Tokenizer class processes raw data by:

- Splitting the input text into individual tokens based on whitespace.
- Building a vocabulary of tokens encountered in the training data.
- Assigning a unique ID to each token in the vocabulary.
- Encoding input text sequences into sequences of token IDs, replacing unknown tokens with a special <unk> token ID and padding sequences to a maximum length.
- Decoding sequences of token IDs back into human-readable text.

The size of the vocabulary corresponds to the number of unique tokens encountered in the training data. In the code, the vocabulary size is dynamically determined during training and is equal to the number of unique tokens added to the vocabulary. The initial vocabulary includes four special tokens (<pad>, <s>, </s>, <unk>), and additional tokens encountered during training are incrementally added to the vocabulary. The vocabulary size is obtained using the get_vocab_size method of the tokenizer instance and it is 23.

b. The maximum length of the input sequence is specified as max_len in the command-line arguments parsed by the argparse.ArgumentParser. Its default value is set to 128, but it can be adjusted by us according to our requirements of the task and dataset.

c. The self-attention mechanism involves several steps:

i. Projection: The input embeddings are projected into query, key, and value vectors. This typically involves linear transformations to project the input embeddings into higher-dimensional spaces suitable for manipulation.
ii. Scoring: Each query vector is compared with all key vectors to produce attention scores. These scores represent the relevance/importance of each key with respect to the query.
iii. Normalization: The attention scores are normalized, usually using softmax, to ensure that the attention weights sum up to 1. This step gives weights to each value vector based on its relevance to the query.
iv. Weighted Sum: The normalized attention scores are used to weight the corresponding value vectors. These weighted vectors are then summed to produce the output of the attention mechanism.
v. Projection Back: The output from the weighted sum step is often projected back to the original embedding dimensionality through another linear transformation.

The critical step to make the self-attention mechanism causal is the masking step. Specifically, this line of code:
attn_weights = attn_weights.masked_fill(self.mask[:, :, :L, :L] == 0, float('-inf'))

This line applies a causal mask to the attention weights, ensuring that each token can only attend to previous tokens or itself, but not to future tokens. This is crucial for models where the generation of each token depends only on the previously generated tokens during training.

The mask in the forward function of the class "GPT" is needed for a couple of reasons:

- Padding Tokens: In sequences, particularly during batch processing, sequences are padded to have uniform lengths. The mask ensures that the attention mechanism does not attend to the padded tokens, which do not contain meaningful information.
- Causal Generation: In autoregressive models like GPT, during generation, we don't want tokens to attend to future tokens. This would violate the autoregressive property. So, the mask ensures that attention is only applied to previous or current tokens, making the generation process causal.

Training process –

The training process involves iterating over multiple epochs, where each epoch consists of iterations over batches of data. Given below is a breakdown of the training process:

- **Initialization:**
    - Load the dataset.
    - Build the tokenizer.
    - Initialize the GPT model and the trainer with the specified configurations.

- **Training Loop:** For each epoch,
    - Start the timer to measure the time taken for the epoch.
    - Run the training phase:
        - Set the model in training mode
        - Create a DataLoader for the training dataset.
        - Iterate over batches:
            - Move the input data to the appropriate device (CPU/GPU)
            - Perform forward pass: Generate logits and calculate the loss
            - Perform backward pass (if in training mode): Zero gradients, Scale the loss using gradient scaling (apex), Backpropagate the loss, Update model parameters, Optionally decay the learning rate based on training progress
            - Update progress and display relevant information.
    - Run the validation phase (if a validation dataset is provided):
        - Set the model in evaluation mode.
        - Create a DataLoader for the validation dataset.
        - Iterate over batches:
            - Move the input data to the appropriate device.
            - Perform forward pass to compute loss.
            - Calculate and record the average loss.
    - Record the training statistics for the epoch, such as training and validation losses.
    - Optionally save the model checkpoint if it improves over the previous best validation loss.
    - Display epoch-level training summary.

  After completing all epochs, print out a results table summarizing the training process, including the architecture configurations, validation loss, and time taken per epoch.

- **Results:** Finally, the script prints a results table summarizing the training process, including the architecture configurations, validation loss, and time taken per epoch.

During training, the script keeps track of the best validation loss and saves the model checkpoint if the validation loss improves. This ensures that the best model is saved for later use. Additionally, learning rate scheduling is performed based on the configured strategy (linear warm-up followed by cosine decay).

d. The generation process involves using the trained GPT model to generate a sequence of actions given a set of conditions. Let us break down the generation process using a concrete example from the test dataset you provided:
Example:
Conditions (IN) : "turn opposite right thrice and turn opposite left"
Expected Actions (OUT): "I_TURN_RIGHT I_TURN_RIGHT I_TURN_RIGHT I_TURN_RIGHT I_TURN_RIGHT I_TURN_RIGHT I_TURN_LEFT I_TURN_LEFT"

(i) Tokenization: The input conditions ("turn opposite right thrice and turn opposite left") are tokenized using the tokenizer. This converts the input text into a sequence of tokens understandable by the model.

(ii) Encoding: The tokenized sequence of conditions is encoded into numerical indices using the tokenizer's vocabulary. This creates a tensor of input IDs representing the conditions.

(iii) Generation Loop:
   a. The model generates the actions iteratively, token by token, starting from the encoded conditions.
   b. At each step, the model predicts the next token in the sequence based on the previously generated tokens.
   c. The predicted token is appended to the input sequence, and the process continues iteratively until the maximum length is reached or a termination condition is met.

(iv) Termination: The generation process terminates when either the maximum length is reached or a predefined stop token is generated.

(v) Decoding: Once the generation process is complete, the generated sequence of action tokens is decoded back into human-readable text using the tokenizer.

(vi) Comparison:
   a. Finally, the generated sequence of actions is compared with the expected actions from the dataset.
   b. If the generated actions match the expected actions, it's considered a correct prediction, and the accuracy is incremented.

In summary, the generation process involves iteratively predicting the next token in the sequence based on the previously generated tokens, until a termination condition is met. This process allows the GPT model to generate sequences of actions conditioned on the provided input conditions.

e. Table

n_layer	n_head	n_embd	epoch	validation loss	time per epoch	Mean Validation loss	Mean time per epoch	Test Accuracy (Generate)
2	**2**	**16**	1	1.3321	8.74	**0.4429**	**7.32**	**0.0662**
2	2	16	2	0.8534	7.41			
2	2	16	3	0.6856	7.11			
2	2	16	4	0.6445	7.34			
2	2	16	5	0.5955	7.26			
2	2	16	6	0.5895	7.37			
2	2	16	7	0.5721	7.32			
2	2	16	8	0.5502	7.25			
2	2	16	9	0.5296	7.41			
2	2	16	10	0.5217	7.44			
2	2	16	11	0.5049	7.36			
2	2	16	12	0.5027	7.04			
2	2	16	13	0.4869	7.16			
2	2	16	14	0.4735	7.14			
2	2	16	15	0.4573	7.48			
2	2	16	16	0.4587	7.49			
2	2	16	17	0.4510	7.28			
2	2	16	18	0.4395	7.15			
2	2	16	19	0.4337	7.22			
2	2	16	20	0.4323	7.34			
2	2	16	21	0.4233	7.41			
2	2	16	22	0.4262	7.15			
2	2	16	23	0.4145	7.21			
2	2	16	24	0.4157	7.32			
2	2	16	25	0.4054	7.16			
2	2	16	26	0.4069	7.23			
2	2	16	27	0.4012	7.18			
2	2	16	28	0.3985	7.05			
2	2	16	29	0.3921	7.31			
2	2	16	30	0.3909	7.13			
2	2	16	31	0.3854	7.20			
2	2	16	32	0.3890	7.36			
2	2	16	33	0.3883	7.33			
2	2	16	34	0.3861	7.37			
2	2	16	35	0.3820	7.36			
2	2	16	36	0.3742	7.30			
2	2	16	37	0.3754	7.26			
2	2	16	38	0.3787	7.24			
2	2	16	39	0.3705	7.43			
2	2	16	40	0.3682	7.44			
2	2	16	41	0.3703	7.17			

2	2	16	42	0.3680	7.27			
2	2	16	43	0.3655	6.97			
2	2	16	44	0.3614	7.33			
2	2	16	45	0.3628	7.14			
2	2	16	46	0.3647	7.42			
2	2	16	47	0.3602	7.46			
2	2	16	48	0.3582	7.68			
2	2	16	49	0.3585	7.34			
2	2	16	50	0.3574	7.32			
2	2	16	51	0.3592	6.99			
2	2	16	52	0.3524	7.33			
2	2	16	53	0.3532	7.29			
2	2	16	54	0.3579	7.37			
2	2	16	55	0.3545	7.50			
2	2	16	56	0.3563	7.44			
2	2	16	57	0.3585	7.33			
2	2	16	58	0.3538	7.40			
2	2	16	59	0.3585	7.31			
2	2	16	60	0.3551	7.30			
2	**4**	**16**	1	1.3210	8.21	**0.3544**	**7.15**	**0.1698**
2	4	16	2	0.8383	7.18			
2	4	16	3	0.6807	7.12			
2	4	16	4	0.6174	7.04			
2	4	16	5	0.5637	7.24			
2	4	16	6	0.5409	7.06			
2	4	16	7	0.5131	6.98			
2	4	16	8	0.4971	7.20			
2	4	16	9	0.4764	7.12			
2	4	16	10	0.4656	7.11			
2	4	16	11	0.4542	7.22			
2	4	16	12	0.4456	7.03			
2	4	16	13	0.4273	7.13			
2	4	16	14	0.4185	7.20			
2	4	16	15	0.4099	7.32			
2	4	16	16	0.3958	7.01			
2	4	16	17	0.3909	7.13			
2	4	16	18	0.3816	7.01			
2	4	16	19	0.3732	7.11			
2	4	16	20	0.3609	7.02			
2	4	16	21	0.3572	7.28			
2	4	16	22	0.3515	7.14			
2	4	16	23	0.3356	7.10			
2	4	16	24	0.3346	6.97			
2	4	16	25	0.3265	6.86			
2	4	16	26	0.3111	7.10			
2	4	16	27	0.3050	7.36			

2	4	16	28	0.2975	7.05			
2	4	16	29	0.2975	7.11			
2	4	16	30	0.2959	6.93			
2	4	16	31	0.2994	7.15			
2	4	16	32	0.2790	7.29			
2	4	16	33	0.2731	7.20			
2	4	16	34	0.2700	6.99			
2	4	16	35	0.2668	7.32			
2	4	16	36	0.2594	7.24			
2	4	16	37	0.2604	7.04			
2	4	16	38	0.2593	7.01			
2	4	16	39	0.2544	7.25			
2	4	16	40	0.2537	7.12			
2	4	16	41	0.2469	7.25			
2	4	16	42	0.2479	7.25			
2	4	16	43	0.2456	7.22			
2	4	16	44	0.2467	7.26			
2	4	16	45	0.2397	7.10			
2	4	16	46	0.2419	7.06			
2	4	16	47	0.2441	7.08			
2	4	16	48	0.2411	7.08			
2	4	16	49	0.2373	7.16			
2	4	16	50	0.2381	7.06			
2	4	16	51	0.2411	6.99			
2	4	16	52	0.2371	7.12			
2	4	16	53	0.2369	7.19			
2	4	16	54	0.2377	7.23			
2	4	16	55	0.2350	7.21			
2	4	16	56	0.2378	7.03			
2	4	16	57	0.2422	7.07			
2	4	16	58	0.2367	7.12			
2	4	16	59	0.2378	7.20			
2	4	16	60	0.2348	7.09			
4	**4**	**32**	**1**	**0.7433**	**11.48**	**0.1000**	**10.25**	**0.9922**
4	4	32	2	0.5521	10.38			
4	4	32	3	0.4576	10.15			
4	4	32	4	0.3655	10.17			
4	4	32	5	0.3162	10.40			
4	4	32	6	0.2680	10.18			
4	4	32	7	0.2353	10.37			
4	4	32	8	0.2098	10.21			
4	4	32	9	0.1876	10.21			
4	4	32	10	0.1745	10.24			
4	4	32	11	0.1463	10.22			
4	4	32	12	0.1332	10.20			
4	4	32	13	0.1151	10.16			

4	4	32	14	0.1044	10.22			
4	4	32	15	0.0885	10.32			
4	4	32	16	0.0888	10.24			
4	4	32	17	0.0769	10.34			
4	4	32	18	0.0740	10.21			
4	4	32	19	0.0711	10.20			
4	4	32	20	0.0684	10.34			
4	4	32	21	0.0995	10.26			
4	4	32	22	0.0680	10.22			
4	4	32	23	0.0746	10.27			
4	4	32	24	0.0484	10.15			
4	4	32	25	0.0826	10.16			
4	4	32	26	0.0524	10.23			
4	4	32	27	0.0530	10.18			
4	4	32	28	0.0448	10.20			
4	4	32	29	0.0364	10.14			
4	4	32	30	0.0408	10.39			
4	4	32	31	0.0431	10.41			
4	4	32	32	0.0325	10.22			
4	4	32	33	0.0516	10.24			
4	4	32	34	0.0384	10.20			
4	4	32	35	0.0444	10.18			
4	4	32	36	0.0351	10.24			
4	4	32	37	0.0416	10.28			
4	4	32	38	0.0378	10.23			
4	4	32	39	0.0362	10.15			
4	4	32	40	0.0330	10.40			
4	4	32	41	0.0320	10.36			
4	4	32	42	0.0266	10.21			
4	4	32	43	0.0354	10.32			
4	4	32	44	0.0291	10.18			
4	4	32	45	0.0255	10.23			
4	4	32	46	0.0277	10.16			
4	4	32	47	0.0239	10.11			
4	4	32	48	0.0306	10.21			
4	4	32	49	0.0262	10.18			
4	4	32	50	0.0249	10.16			
4	4	32	51	0.0263	10.19			
4	4	32	52	0.0264	10.16			
4	4	32	53	0.0262	10.28			
4	4	32	54	0.0218	10.21			
4	4	32	55	0.0207	10.17			
4	4	32	56	0.0254	10.20			
4	4	32	57	0.0236	10.34			
4	4	32	58	0.0322	10.14			
4	4	32	59	0.0218	10.30			

4	4	32	60	0.0260	10.16			
6	**8**	**64**	**1**	**0.5135**	**15.27**	**0.0432**	**14.49**	**0.9931**
6	8	64	2	0.3070	13.81			
6	8	64	3	0.2427	14.17			
6	8	64	4	0.1866	14.35			
6	8	64	5	0.1432	14.50			
6	8	64	6	0.1127	14.55			
6	8	64	7	0.0858	14.66			
6	8	64	8	0.1301	14.64			
6	8	64	9	0.0524	14.44			
6	8	64	10	0.0673	14.63			
6	8	64	11	0.0539	14.49			
6	8	64	12	0.0458	14.62			
6	8	64	13	0.0420	14.42			
6	8	64	14	0.0357	14.53			
6	8	64	15	0.0347	14.39			
6	8	64	16	0.0279	14.55			
6	8	64	17	0.0238	14.54			
6	8	64	18	0.0291	14.68			
6	8	64	19	0.0278	14.50			
6	8	64	20	0.0199	14.51			
6	8	64	21	0.0304	14.24			
6	8	64	22	0.0260	14.46			
6	8	64	23	0.0262	14.67			
6	8	64	24	0.0209	14.09			
6	8	64	25	0.0228	14.27			
6	8	64	26	0.0144	14.17			
6	8	64	27	0.0130	14.51			
6	8	64	28	0.0132	14.29			
6	8	64	29	0.0130	14.51			
6	8	64	30	0.0117	14.50			
6	8	64	31	0.0163	14.73			
6	8	64	32	0.0150	14.49			
6	8	64	33	0.0081	14.59			
6	8	64	34	0.0082	14.75			
6	8	64	35	0.0108	14.57			
6	8	64	36	0.0210	14.65			
6	8	64	37	0.0062	14.64			
6	8	64	38	0.0119	14.69			
6	8	64	39	0.0111	14.68			
6	8	64	40	0.0071	14.17			
6	8	64	41	0.0074	14.41			
6	8	64	42	0.0101	14.35			
6	8	64	43	0.0035	14.34			
6	8	64	44	0.0093	14.32			
6	8	64	45	0.0129	14.36			

6	8	64	46	0.0027	14.66			
6	8	64	47	0.0046	14.32			
6	8	64	48	0.0064	14.70			
6	8	64	49	0.0051	14.47			
6	8	64	50	0.0045	14.61			
6	8	64	51	0.0042	14.53			
6	8	64	52	0.0022	14.43			
6	8	64	53	0.0034	14.55			
6	8	64	54	0.0060	14.56			
6	8	64	55	0.0054	14.52			
6	8	64	56	0.0032	14.57			
6	8	64	57	0.0053	14.52			
6	8	64	58	0.0024	14.56			
6	8	64	59	0.0025	14.38			
6	8	64	60	0.0017	14.38			
8	**8**	**128**	**1**	**0.4003**	**18.47**	**0.0257**	**17.02**	**1.0000**
8	8	128	2	0.2010	16.97			
8	8	128	3	0.1376	17.19			
8	8	128	4	0.1069	16.81			
8	8	128	5	0.1006	16.93			
8	8	128	6	0.0845	17.13			
8	8	128	7	0.0629	16.98			
8	8	128	8	0.0692	17.09			
8	8	128	9	0.0350	16.83			
8	8	128	10	0.0497	17.09			
8	8	128	11	0.0297	16.78			
8	8	128	12	0.0314	16.96			
8	8	128	13	0.0376	16.81			
8	8	128	14	0.0168	17.08			
8	8	128	15	0.0186	16.86			
8	8	128	16	0.0146	16.68			
8	8	128	17	0.0157	16.86			
8	8	128	18	0.0152	17.12			
8	8	128	19	0.0225	17.01			
8	8	128	20	0.0074	17.37			
8	8	128	21	0.0064	16.90			
8	8	128	22	0.0082	16.81			
8	8	128	23	0.0068	16.86			
8	8	128	24	0.0037	16.84			
8	8	128	25	0.0106	16.92			
8	8	128	26	0.0040	16.89			
8	8	128	27	0.0029	16.90			
8	8	128	28	0.0035	17.06			
8	8	128	29	0.0011	17.11			
8	8	128	30	0.0019	16.82			
8	8	128	31	0.0032	16.90			

8	8	128	32	0.0009	16.77			
8	8	128	33	0.0018	17.10			
8	8	128	34	0.0037	16.96			
8	8	128	35	0.0074	16.90			
8	8	128	36	0.0067	17.07			
8	8	128	37	0.0044	17.47			
8	8	128	38	0.0005	17.06			
8	8	128	39	0.0016	17.17			
8	8	128	40	0.0005	17.11			
8	8	128	41	0.0007	17.07			
8	8	128	42	0.0001	16.94			
8	8	128	43	0.0002	17.07			
8	8	128	44	0.0014	16.82			
8	8	128	45	0.0001	17.00			
8	8	128	46	0.0001	17.28			
8	8	128	47	0.0003	17.12			
8	8	128	48	0.0004	16.77			
8	8	128	49	0.0001	16.97			
8	8	128	50	0.0000	16.94			
8	8	128	51	0.0008	17.05			
8	8	128	52	0.0001	16.98			
8	8	128	53	0.0001	16.94			
8	8	128	54	0.0001	17.01			
8	8	128	55	0.0010	17.46			
8	8	128	56	0.0000	17.17			
8	8	128	57	0.0001	16.82			
8	8	128	58	0.0000	16.93			
8	8	128	59	0.0001	17.07			
8	8	128	60	0.0000	17.08			
10	**16**	**256**	**1**	**0.3069**	**21.43**	**0.0170**	**20.64**	**1.0000**
10	16	256	2	0.1537	19.80			
10	16	256	3	0.1245	20.25			
10	16	256	4	0.0822	20.18			
10	16	256	5	0.0406	20.24			
10	16	256	6	0.0389	20.09			
10	16	256	7	0.0594	20.68			
10	16	256	8	0.0229	20.69			
10	16	256	9	0.0234	20.64			
10	16	256	10	0.0277	20.96			
10	16	256	11	0.0146	20.61			
10	16	256	12	0.0242	20.63			
10	16	256	13	0.0107	20.57			
10	16	256	14	0.0116	20.46			
10	16	256	15	0.0136	20.57			
10	16	256	16	0.0068	20.64			
10	16	256	17	0.0051	20.58			

10	16	256	18	0.0055	20.84			
10	16	256	19	0.0056	20.57			
10	16	256	20	0.0052	20.38			
10	16	256	21	0.0059	21.00			
10	16	256	22	0.0038	20.70			
10	16	256	23	0.0025	20.72			
10	16	256	24	0.0021	20.69			
10	16	256	25	0.0025	21.17			
10	16	256	26	0.0015	20.77			
10	16	256	27	0.0011	20.86			
10	16	256	28	0.0020	20.80			
10	16	256	29	0.0009	20.62			
10	16	256	30	0.0009	20.38			
10	16	256	31	0.0016	20.47			
10	16	256	32	0.0006	21.02			
10	16	256	33	0.0015	20.24			
10	16	256	34	0.0009	20.87			
10	16	256	35	0.0028	20.66			
10	16	256	36	0.0005	21.09			
10	16	256	37	0.0002	21.14			
10	16	256	38	0.0003	20.68			
10	16	256	39	0.0009	20.64			
10	16	256	40	0.0003	20.52			
10	16	256	41	0.0003	20.87			
10	16	256	42	0.0002	20.96			
10	16	256	43	0.0003	20.70			
10	16	256	44	0.0004	20.61			
10	16	256	45	0.0002	20.77			
10	16	256	46	0.0002	20.69			
10	16	256	47	0.0002	20.46			
10	16	256	48	0.0004	20.98			
10	16	256	49	0.0003	20.69			
10	16	256	50	0.0003	20.40			
10	16	256	51	0.0003	20.38			
10	16	256	52	0.0000	20.60			
10	16	256	53	0.0002	20.91			
10	16	256	54	0.0002	20.52			
10	16	256	55	0.0001	20.80			
10	16	256	56	0.0002	20.49			
10	16	256	57	0.0001	20.32			
10	16	256	58	0.0004	20.35			
10	16	256	59	0.0003	20.55			
10	16	256	60	0.0002	20.48			

Based on the provided data, we can observe the impact of the hyperparameters (number of layers, number of heads, and embedding size) on the model performance:

- Number of Layers (n_layer): As the number of layers increases, the model tends to perform better in terms of mean validation loss and test accuracy. This is evident from the trend where the mean validation loss decreases as the number of layers increases. However, increasing the number of layers also leads to an increase in the time per epoch, as more computations are required for training.
- Number of Heads (n_head): Increasing the number of heads generally leads to improved performance, as seen in the decreasing trend of mean validation loss and increasing trend of test accuracy. However, the improvement is not as pronounced as increasing the number of layers.
- Embedding Size (n_embd): Larger embedding sizes tend to result in better performance, as indicated by the decreasing trend in mean validation loss and increasing trend in test accuracy. However, larger embedding sizes also require more computational resources and may increase training time.

So, increasing the number of layers, heads, and embedding size generally improves model performance in terms of validation loss and test accuracy. However, this improvement comes at the cost of increased computational complexity and training time. Therefore, there is a trade-off between model performance and computational resources.

f. Used the data split **'length'**. The split is designed to evaluate the model's ability to generalize and produce accurate outputs for longer action sequences that contain combinations of familiar elements (verbs, modifiers, and conjunctions) seen during training but in novel combinations or lengths. This evaluation assesses the model's capacity to effectively combine and extend learned actions and linguistic structures to generate coherent and contextually appropriate sequences, even when faced with previously unseen combinations or lengths of actions.

n_layer	n_head	n_embd	validation loss	time per epoch	mean validation loss	mean time per epoch	Testing Accuracy (Generate)
50	**2**	**16**	1.3816	97.57	**0.1599**	**85.75**	**0.00**
50	2	16	1.0269	86.88			
50	2	16	0.8456	86.89			
50	2	16	0.7739	87.82			
50	2	16	0.7046	87.01			
50	2	16	0.6337	87.13			
50	2	16	0.5708	87.60			
50	2	16	0.4480	86.94			
50	2	16	0.3884	87.45			

50	2	16	0.3069	87.30			
50	2	16	0.2482	87.28			
50	2	16	0.2058	87.34			
50	2	16	0.1696	87.22			
50	2	16	0.1351	87.16			
50	2	16	0.1160	87.07			
50	2	16	0.1353	86.49			
50	2	16	0.0813	86.01			
50	2	16	0.1120	87.33			
50	2	16	0.0697	87.22			
50	2	16	0.0635	87.53			
50	2	16	0.1273	86.09			
50	2	16	0.0633	86.77			
50	2	16	0.0598	83.87			
50	2	16	0.0471	83.62			
50	2	16	0.0432	83.41			
50	2	16	0.0627	86.62			
50	2	16	0.0622	85.15			
50	2	16	0.0428	86.73			
50	2	16	0.0350	85.15			
50	2	16	0.0313	85.83			
50	2	16	0.0344	84.07			
50	2	16	0.0369	85.96			
50	2	16	0.0296	86.35			
50	2	16	0.0283	85.82			
50	2	16	0.0478	86.16			
50	2	16	0.0235	85.60			
50	2	16	0.0353	82.26			
50	2	16	0.0209	82.60			
50	2	16	0.0211	83.15			
50	2	16	0.0189	85.60			
50	2	16	0.0173	85.44			
50	2	16	0.0182	86.89			
50	2	16	0.0171	85.25			
50	2	16	0.0163	85.59			
50	2	16	0.0159	87.78			
50	2	16	0.0161	91.30			
50	2	16	0.0151	86.03			
50	2	16	0.0155	85.81			
50	2	16	0.0172	85.84			
50	2	16	0.0151	86.19			
50	2	16	0.0132	85.80			
50	2	16	0.0181	82.20			
50	2	16	0.0137	82.28			
50	2	16	0.0151	82.62			

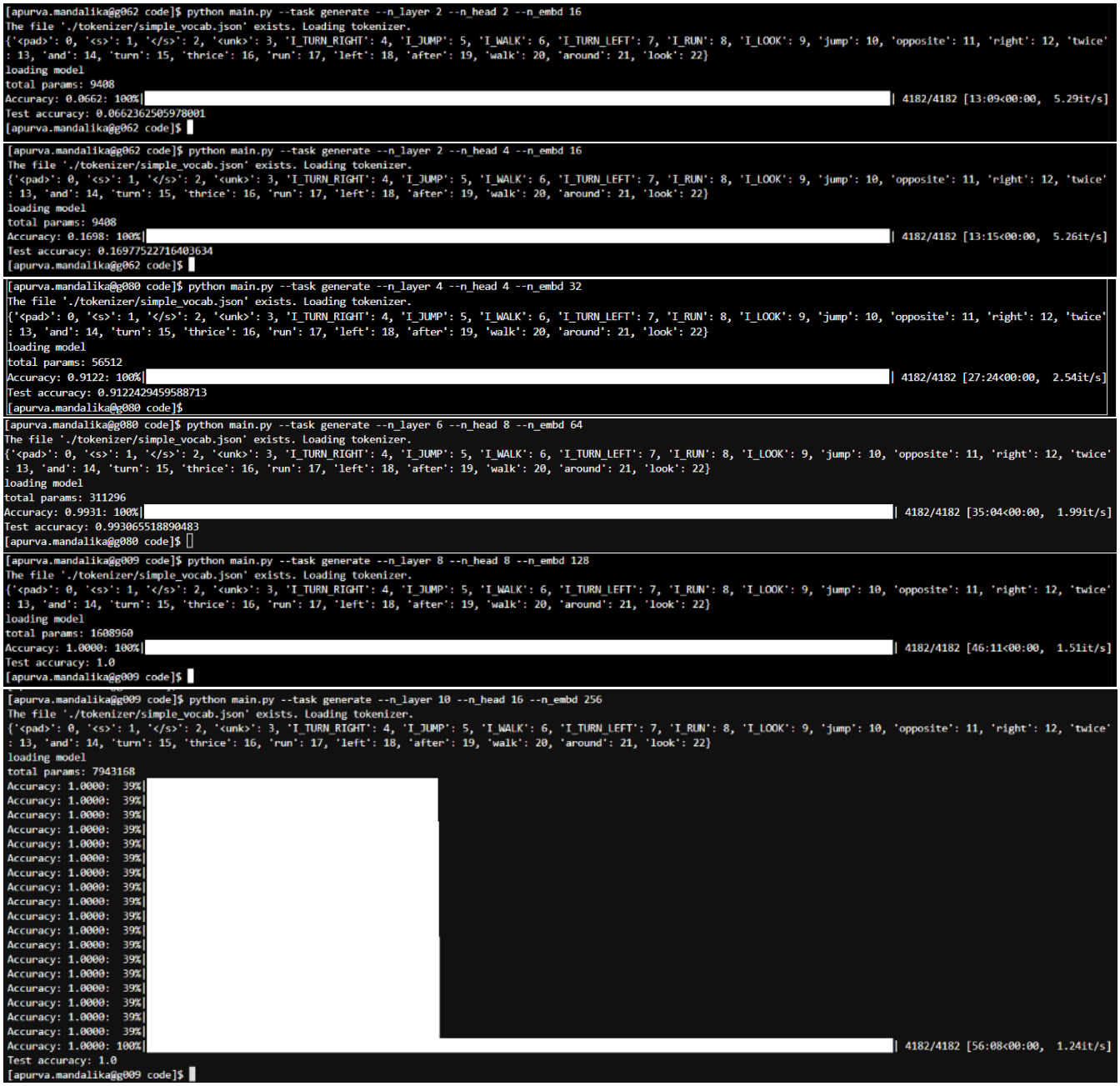
50	2	16	0.0146	81.54			
50	2	16	0.0129	82.08			
50	2	16	0.0147	81.85			
50	2	16	0.0139	81.74			
50	2	16	0.0129	82.06			
50	**2**	**16**	**0.0138**	82.79			

Insights:

- Our CSA based GPT model, by design, imposes causality constraints on the attention mechanism, ensuring that each token attends only to previous tokens or itself. This is particularly useful for autoregressive tasks where the generation of each token depends only on previously generated tokens. On the other hand, the attention mechanism used in the GRU model might be more flexible and adaptive, allowing the model to attend to any part of the input sequence based on its relevance to the current context. This flexibility could be advantageous, especially for tasks with longer input sequences.
- Our model, offers high model capacity with its multiple layers, however, the ability of CSA to generalize to longer sequences might be limited compared to models equipped with more flexible attention mechanisms. The GRU model mentioned in the paper, despite having a smaller capacity, achieves competitive performance by effectively leveraging attention mechanisms, indicating the importance of attention in handling complex sequence modelling tasks.

Appendix

- ‘Simple’ Data Split - Test Accuracies



- ‘length’ Data Split – Train loss, Validation loss and Test Accuracies

```
epoch 15 iter 477: train loss 0.27436. lr 3.4284e-04: 100%| | 478/478 [01:23<00:00,  5.69it/s]
test loss: %f 0.11595657950750103
epoch_valid_loss: 0.11595657950750103, epoch_train_loss: 0.2738531071334204, epoch: 15
Saving at epoch 15: ./cond_gpt/weights/None_lengthsplit_50layer_2head_16embd_32bs.pt
step_train_loss: 0.25393548607826233 train_step: 7500, learning_rate: 0.00033770187882238545 | 330/478 [00:58<00:25,  5.74it/s]
epoch 16 iter 477: train loss 0.20920. lr 3.3536e-04: 100%| | 478/478 [01:23<00:00,  5.73it/s]
test loss: %f 0.1352660658734816
epoch_valid_loss: 0.1352660658734816, epoch_train_loss: 0.2505001943542868, epoch: 16
step_train_loss: 0.17839477956295013 train_step: 8000, learning_rate: 0.00032960651359957967 | 352/478 [01:00<00:22,  5.70it/s]
epoch 17 iter 477: train loss 0.25633. lr 3.2752e-04: 100%| | 478/478 [01:22<00:00,  5.77it/s]
test loss: %f 0.08133871956831878
epoch_valid_loss: 0.08133871956831878, epoch_train_loss: 0.22959262999281224, epoch: 17
Saving at epoch 17: ./cond_gpt/weights/None_lengthsplit_50layer_2head_16embd_32bs.pt
step_train_loss: 0.21427012979984283 train_step: 8500, learning_rate: 0.0003211272206482685 | 374/478 [01:06<00:18,  5.70it/s]
epoch 18 iter 477: train loss 0.19505. lr 3.1934e-04: 100%| | 478/478 [01:24<00:00,  5.67it/s]
test loss: %f 0.11203097965982226
epoch_valid_loss: 0.11203097965982226, epoch_train_loss: 0.21735326581046172, epoch: 18
step_train_loss: 0.20009490847587585 train_step: 9000, learning_rate: 0.00031228911780853354 | 396/478 [01:09<00:14,  5.72it/s]
epoch 19 iter 477: train loss 0.21242. lr 3.1083e-04: 100%| | 478/478 [01:24<00:00,  5.68it/s]
test loss: %f 0.06970568711834925
epoch_valid_loss: 0.06970568711834925, epoch_train_loss: 0.1983176127200845, epoch: 19
Saving at epoch 19: ./cond_gpt/weights/None_lengthsplit_50layer_2head_16embd_32bs.pt
step_train_loss: 0.2263641357421875 train_step: 9500, learning_rate: 0.0003031183674392653 | 418/478 [01:13<00:10,  5.74it/s]
epoch 20 iter 477: train loss 0.13543. lr 3.0202e-04: 100%| | 478/478 [01:24<00:00,  5.66it/s]
test loss: %f 0.06352007120019859
epoch_valid_loss: 0.06352007120019859, epoch_train_loss: 0.1882161243378368, epoch: 20
Saving at epoch 20: ./cond_gpt/weights/None_lengthsplit_50layer_2head_16embd_32bs.pt
step_train_loss: 0.12573248147964478 train_step: 10000, learning_rate: 0.00029364217176236026 | 440/478 [01:16<00:06,  6.05it/s]
epoch 21 iter 477: train loss 0.13359. lr 2.9293e-04: 100%| | 478/478 [01:22<00:00,  5.76it/s]
test loss: %f 0.12730575766828325
epoch_valid_loss: 0.12730575766828325, epoch_train_loss: 0.1779476337010142, epoch: 21
step_train_loss: 0.13674236834049225 train_step: 10500, learning_rate: 0.00028388885839598702 | 462/478 [01:21<00:02,  6.03it/s]
epoch 22 iter 477: train loss 0.16842. lr 2.8359e-04: 100%| | 478/478 [01:23<00:00,  5.71it/s]
test loss: %f 0.06330315358246918
epoch_valid_loss: 0.06330315358246918, epoch_train_loss: 0.16531385019982708, epoch: 22
Saving at epoch 22: ./cond_gpt/weights/None_lengthsplit_50layer_2head_16embd_32bs.pt
epoch 23 iter 477: train loss 0.13672. lr 2.7403e-04: 100%| | 478/478 [01:20<00:00,  5.92it/s]
test loss: %f 0.05981088084755121
```

```
step_train_loss: 0.12909094989299774 train_step: 11500, learning_rate: 0.0002636687494788558 | 28/478 [00:04<01:15,  5.97it/s]
epoch 25 iter 477: train loss 0.12187. lr 2.5433e-04: 100%| | 478/478 [01:20<00:00,  5.95it/s]
test loss: %f 0.04317793250083923
epoch_valid_loss: 0.04317793250083923, epoch_train_loss: 0.14064275197466547, epoch: 25
Saving at epoch 25: ./cond_gpt/weights/None_lengthsplit_50layer_2head_16embd_32bs.pt
step_train_loss: 0.1335783749818802 train_step: 12000, learning_rate: 0.00025325925168453116 | 50/478 [00:08<01:11,  5.99it/s]
epoch 26 iter 477: train loss 0.14351. lr 2.4424e-04: 100%| | 478/478 [01:23<00:00,  5.72it/s]
test loss: %f 0.06270570887459649
epoch_valid_loss: 0.06270570887459649, epoch_train_loss: 0.1356973075872934, epoch: 26
step_train_loss: 0.11145858466625214 train_step: 12500, learning_rate: 0.00024269198632935544 | 72/478 [00:12<01:11,  5.65it/s]
epoch 27 iter 477: train loss 0.12624. lr 2.3404e-04: 100%| | 478/478 [01:22<00:00,  5.83it/s]
test loss: %f 0.06220773842047762
epoch_valid_loss: 0.06220773842047762, epoch_train_loss: 0.12931388097515167, epoch: 27
step_train_loss: 0.13884437084197998 train_step: 13000, learning_rate: 0.00023199823520207903 | 94/478 [00:15<01:05,  5.91it/s]
epoch 28 iter 477: train loss 0.18461. lr 2.2374e-04: 100%| | 478/478 [01:23<00:00,  5.72it/s]
test loss: %f 0.042814590554270476
epoch_valid_loss: 0.042814590554270476, epoch_train_loss: 0.12242183658181374, epoch: 28
Saving at epoch 28: ./cond_gpt/weights/None_lengthsplit_50layer_2head_16embd_32bs.pt
step_train_loss: 0.17768850922584534 train_step: 13500, learning_rate: 0.00022120971803984038 | 116/478 [00:20<01:04,  5.64it/s]
epoch 29 iter 477: train loss 0.13608. lr 2.1338e-04: 100%| | 478/478 [01:22<00:00,  5.82it/s]
test loss: %f 0.0350393446645251
epoch_valid_loss: 0.0350393446645251, epoch_train_loss: 0.11676899780944053, epoch: 29
Saving at epoch 29: ./cond_gpt/weights/None_lengthsplit_50layer_2head_16embd_32bs.pt
step_train_loss: 0.10659568011760712 train_step: 14000, learning_rate: 0.00021035837225392284 | 138/478 [00:23<01:01,  5.53it/s]
epoch 30 iter 477: train loss 0.12972. lr 2.0298e-04: 100%| | 478/478 [01:22<00:00,  5.78it/s]
test loss: %f 0.031282561727695994
epoch_valid_loss: 0.031282561727695994, epoch_train_loss: 0.1132709459171759, epoch: 30
Saving at epoch 30: ./cond_gpt/weights/None_lengthsplit_50layer_2head_16embd_32bs.pt
step_train_loss: 0.15128064155578613 train_step: 14500, learning_rate: 0.00019947632087609078 | 160/478 [00:26<00:52,  6.07it/s]
epoch 31 iter 477: train loss 0.08642. lr 1.9258e-04: 100%| | 478/478 [01:20<00:00,  5.91it/s]
test loss: %f 0.03438812578786855
epoch_valid_loss: 0.03438812578786855, epoch_train_loss: 0.10855516615990066, epoch: 31
step_train_loss: 0.09286806732416153 train_step: 15000, learning_rate: 0.00018859584217847975 | 182/478 [00:31<00:51,  5.76it/s]
epoch 32 iter 477: train loss 0.09909. lr 1.8219e-04: 100%| | 478/478 [01:22<00:00,  5.77it/s]
test loss: %f 0.03687326258255376
epoch_valid_loss: 0.03687326258255376, epoch_train_loss: 0.10515929033753763, epoch: 32
step_train_loss: 0.07445827126502991 train_step: 15500, learning_rate: 0.00017774914552075945 | 204/478 [00:35<00:47,  5.77it/s]
epoch 33 iter 477: train loss 0.12933. lr 1.7185e-04: 100%| | 478/478 [01:23<00:00,  5.74it/s]
```

```
test loss: %f 0.029598432248113333
epoch_valid_loss: 0.029598432248113333, epoch_train_loss: 0.10103700118271876, epoch: 33
Saving at epoch 33: ./cond_gpt/weights/None_lengthsplit_50layer_2head_16embd_32bs.pt
step_train_loss: 0.08311779797077179 train_step: 16000, learning_rate: 0.00016696836159981425 | 226/478 [00:39<00:43,  5.80it/s]
epoch 34 iter 477: train loss 0.10155. lr 1.6159e-04: 100%| | 478/478 [01:22<00:00,  5.78it/s]
test loss: %f 0.028342962644442363
epoch_valid_loss: 0.028342962644442363, epoch_train_loss: 0.09698281119184016, epoch: 34
Saving at epoch 34: ./cond_gpt/weights/None_lengthsplit_50layer_2head_16embd_32bs.pt
step_train_loss: 0.1051926240324974 train_step: 16500, learning_rate: 0.00015628542586217074 | 248/478 [00:43<00:40,  5.73it/s]
epoch 35 iter 477: train loss 0.06661. lr 1.5144e-04: 100%| | 478/478 [01:23<00:00,  5.75it/s]
test loss: %f 0.04783162883379393
epoch_valid_loss: 0.04783162883379393, epoch_train_loss: 0.09303848630121313, epoch: 35
step_train_loss: 0.09395767003297806 train_step: 17000, learning_rate: 0.00014573198390292774 | 270/478 [00:46<00:35,  5.82it/s]
epoch 36 iter 477: train loss 0.07351. lr 1.4141e-04: 100%| | 478/478 [01:22<00:00,  5.79it/s]
test loss: %f 0.02354517739473118
epoch_valid_loss: 0.02354517739473118, epoch_train_loss: 0.09020787828200531, epoch: 36
Saving at epoch 36: ./cond_gpt/weights/None_lengthsplit_50layer_2head_16embd_32bs.pt
step_train_loss: 0.08678682148456573 train_step: 17500, learning_rate: 0.00013533933180092872 | 292/478 [00:48<00:30,  6.11it/s]
epoch 37 iter 477: train loss 0.10010. lr 1.3154e-04: 100%| | 478/478 [01:19<00:00,  6.04it/s]
test loss: %f 0.03534271942313622
epoch_valid_loss: 0.03534271942313622, epoch_train_loss: 0.08673832455657765, epoch: 37
step_train_loss: 0.08914818614721298 train_step: 18000, learning_rate: 0.00012513815312508355 | 314/478 [00:52<00:26,  6.09it/s]
epoch 38 iter 477: train loss 0.06940. lr 1.2186e-04: 100%| | 478/478 [01:19<00:00,  6.01it/s]
test loss: %f 0.02087155810591799
epoch_valid_loss: 0.02087155810591799, epoch_train_loss: 0.0823182707816113, epoch: 38
Saving at epoch 38: ./cond_gpt/weights/None_lengthsplit_50layer_2head_16embd_32bs.pt
step_train_loss: 0.05500321090221405 train_step: 18500, learning_rate: 0.00011515874910140612 | 336/478 [00:55<00:23,  6.09it/s]
epoch 39 iter 477: train loss 0.06567. lr 1.1239e-04: 100%| | 478/478 [01:20<00:00,  5.97it/s]
test loss: %f 0.021083436544156738
epoch_valid_loss: 0.021083436544156738, epoch_train_loss: 0.08251159249660735, epoch: 39
step_train_loss: 0.06773664802312851 train_step: 19000, learning_rate: 0.00010543070538687811 | 358/478 [01:01<00:20,  5.77it/s]
epoch 40 iter 477: train loss 0.09866. lr 1.0316e-04: 100%| | 478/478 [01:22<00:00,  5.79it/s]
test loss: %f 0.01885422461028038 5
epoch_valid_loss: 0.01885422461028038 5, epoch_train_loss: 0.07876986779163447, epoch: 40
Saving at epoch 40: ./cond_gpt/weights/None_lengthsplit_50layer_2head_16embd_32bs.pt
step_train_loss: 0.0539272166788578 train_step: 19500, learning_rate: 9.598276245103928e-05 | 380/478 [01:05<00:16,  5.78it/s]
epoch 41 iter 477: train loss 0.06956. lr 9.4187e-05: 100%| | 478/478 [01:22<00:00,  5.80it/s]
test loss: %f 0.01730320260308131
```

```
step_train_loss: 0.0539272166788578 train_step: 19500, learning_rate: 9.598276245103928e-05 | 380/478 [01:05<00:16,  5.78it/s]
epoch 41 iter 477: train loss 0.06956. lr 9.4187e-05: 100%| | 478/478 [01:22<00:00,  5.80it/s]
test loss: %f 0.01730320260308131
epoch_valid_loss: 0.01730320260308131, epoch_train_loss: 0.07846453028368651, epoch: 41
Saving at epoch 41: ./cond_gpt/weights/None_lengthsplit_50layer_2head_16embd_32bs.pt
step_train_loss: 0.0832921713590622 train_step: 20000, learning_rate: 8.684294645931604e-05 | 402/478 [01:10<00:13,  5.65it/s]
epoch 42 iter 477: train loss 0.06405. lr 8.5503e-05: 100%| | 478/478 [01:23<00:00,  5.70it/s]
test loss: %f 0.018228005781700765
epoch_valid_loss: 0.018228005781700765, epoch_train_loss: 0.07596003208075607, epoch: 42
step_train_loss: 0.06063132360577583 train_step: 20500, learning_rate: 7.803836597700791e-05 | 424/478 [01:12<00:08,  6.26it/s]
epoch 43 iter 477: train loss 0.07765. lr 7.7128e-05: 100%| | 478/478 [01:22<00:00,  5.82it/s]
test loss: %f 0.017130475913829827
epoch_valid_loss: 0.017130475913829827, epoch_train_loss: 0.07145554503761076, epoch: 43
Saving at epoch 43: ./cond_gpt/weights/None_lengthsplit_50layer_2head_16embd_32bs.pt
step_train_loss: 0.0774172767996788 train_step: 21000, learning_rate: 6.959503319984364e-05 | 446/478 [01:16<00:05,  5.42it/s]
epoch 44 iter 477: train loss 0.04170. lr 6.9086e-05: 100%| | 478/478 [01:22<00:00,  5.79it/s]
test loss: %f 0.016327320253131567
epoch_valid_loss: 0.016327320253131567, epoch_train_loss: 0.07030916213599583, epoch: 44
Saving at epoch 44: ./cond_gpt/weights/None_lengthsplit_50layer_2head_16embd_32bs.pt
step_train_loss: 0.08147433400154114 train_step: 21500, learning_rate: 6.153799489525387e-05 | 468/478 [01:21<00:01,  5.84it/s]
epoch 45 iter 477: train loss 0.03274. lr 6.1399e-05: 100%| | 478/478 [01:23<00:00,  5.73it/s]
test loss: %f 0.0159001647552941
epoch_valid_loss: 0.0159001647552941, epoch_train_loss: 0.06880693894394023, epoch: 45
Saving at epoch 45: ./cond_gpt/weights/None_lengthsplit_50layer_2head_16embd_32bs.pt
epoch 46 iter 477: train loss 0.05859. lr 5.4087e-05: 100%| | 478/478 [01:28<00:00,  5.42it/s]
test loss: %f 0.016065317162967944
epoch_valid_loss: 0.016065317162967944, epoch_train_loss: 0.0688889767374516, epoch: 46
step_train_loss: 0.060577377676963806 train_step: 22000, learning_rate: 5.38934603823068e-05 | 12/478 [00:02<01:20,  5.78it/s]
epoch 47 iter 477: train loss 0.05323. lr 4.7170e-05: 100%| | 478/478 [01:22<00:00,  5.77it/s]
test loss: %f 0.015067671697276333
epoch_valid_loss: 0.015067671697276333, epoch_train_loss: 0.06667322512700467, epoch: 47
Saving at epoch 47: ./cond_gpt/weights/None_lengthsplit_50layer_2head_16embd_32bs.pt
step_train_loss: 0.08983886986970901 train_step: 22500, learning_rate: 4.6679285071265955e-05 | 34/478 [00:05<01:16,  5.80it/s]
epoch 48 iter 477: train loss 0.06861. lr 4.0667e-05: 100%| | 478/478 [01:22<00:00,  5.78it/s]
test loss: %f 0.015491769450751168
epoch_valid_loss: 0.015491769450751168, epoch_train_loss: 0.06459166876052089, epoch: 48
step_train_loss: 0.09428656101226807 train_step: 23000, learning_rate: 4e-05 | 56/478 [00:09<01:12,  5.80it/s]
epoch 49 iter 477: train loss 0.04646. lr 4.0000e-05: 100%| | 478/478 [01:22<00:00,  5.78it/s]
```

```
step_train_loss: 0.09428656101226807 train_step: 23000, learning_rate: 4e-05                                  | 56/478 [00:09<01:12,  5.80it/s]
epoch 49 iter 477: train loss 0.04646. lr 4.0000e-05: 100%|                                                   | 478/478 [01:22<00:00,  5.78it/s]
test loss: %f 0.017191762432318042
epoch_valid_loss: 0.017191762432318042, epoch_train_loss: 0.06294228218760825, epoch: 49
step_train_loss: 0.04153740033507347 train_step: 23500, learning_rate: 4e-05                                  | 78/478 [00:13<01:09,  5.79it/s]
epoch 50 iter 477: train loss 0.05615. lr 4.0000e-05: 100%|                                                   | 478/478 [01:23<00:00,  5.75it/s]
test loss: %f 0.015099257135901737
epoch_valid_loss: 0.015099257135901737, epoch_train_loss: 0.06510838146114947, epoch: 50
step_train_loss: 0.06144499033689499 train_step: 24000, learning_rate: 4e-05                                  | 100/478 [00:17<01:05,  5.76it/s]
epoch 51 iter 477: train loss 0.14123. lr 4.0000e-05: 100%|                                                   | 478/478 [01:22<00:00,  5.78it/s]
test loss: %f 0.013171155885275867
epoch_valid_loss: 0.013171155885275867, epoch_train_loss: 0.063196132692146, epoch: 51
Saving at epoch 51: ./cond_gpt/weights/None_lengthsplit_50layer_2head_16embd_32bs.pt
step_train_loss: 0.055662546306848526 train_step: 24500, learning_rate: 4e-05                                 | 122/478 [00:20<00:59,  6.01it/s]
epoch 52 iter 477: train loss 0.03850. lr 4.0000e-05: 100%|                                                   | 478/478 [01:19<00:00,  6.05it/s]
test loss: %f 0.018089427785189065
epoch_valid_loss: 0.018089427785189065, epoch_train_loss: 0.06317551404814466, epoch: 52
step_train_loss: 0.06107360124588013 train_step: 25000, learning_rate: 4e-05                                  | 144/478 [00:23<00:55,  6.06it/s]
epoch 53 iter 477: train loss 0.06003. lr 4.0000e-05: 100%|                                                   | 478/478 [01:19<00:00,  6.03it/s]
test loss: %f 0.013713264096252344
epoch_valid_loss: 0.013713264096252344, epoch_train_loss: 0.061785944523172896, epoch: 53
step_train_loss: 0.044118534773588 train_step: 25500, learning_rate: 4e-05                                    | 166/478 [00:28<00:51,  6.10it/s]
epoch 54 iter 477: train loss 0.08961. lr 4.0000e-05: 100%|                                                   | 478/478 [01:19<00:00,  6.01it/s]
test loss: %f 0.015097079953799645
epoch_valid_loss: 0.015097079953799645, epoch_train_loss: 0.06042488103257063, epoch: 54
step_train_loss: 0.12159059941768646 train_step: 26000, learning_rate: 4e-05                                  | 188/478 [00:30<00:47,  6.17it/s]
epoch 55 iter 477: train loss 0.17466. lr 4.0000e-05: 100%|                                                   | 478/478 [01:18<00:00,  6.09it/s]
test loss: %f 0.014583172395187258
epoch_valid_loss: 0.014583172395187258, epoch_train_loss: 0.06130679169178757, epoch: 55
step_train_loss: 0.05516187846660614 train_step: 26500, learning_rate: 4e-05                                  | 210/478 [00:34<00:44,  6.09it/s]
epoch 56 iter 477: train loss 0.03525. lr 4.0000e-05: 100%|                                                   | 478/478 [01:18<00:00,  6.05it/s]
test loss: %f 0.012903952738270164
epoch_valid_loss: 0.012903952738270164, epoch_train_loss: 0.06088264544093085, epoch: 56
Saving at epoch 56: ./cond_gpt/weights/None_lengthsplit_50layer_2head_16embd_32bs.pt
step_train_loss: 0.10091575980186462 train_step: 27000, learning_rate: 4e-05                                  | 232/478 [00:38<00:40,  6.13it/s]
epoch 57 iter 477: train loss 0.05240. lr 4.0000e-05: 100%|                                                   | 478/478 [01:18<00:00,  6.07it/s]
test loss: %f 0.014689087841866745
epoch_valid_loss: 0.014689087841866745, epoch_train_loss: 0.0604975133312079, epoch: 57
```

```
Saving at epoch 56: ./cond_gpt/weights/None_lengthsplit_50layer_2head_16embd_32bs.pt
step_train_loss: 0.10091575980186462 train_step: 27000, learning_rate: 4e-05                                  | 232/478 [00:38<00:40,  6.13it/s]
epoch 57 iter 477: train loss 0.05240. lr 4.0000e-05: 100%|                                                   | 478/478 [01:18<00:00,  6.07it/s]
test loss: %f 0.014689087841866745
epoch_valid_loss: 0.014689087841866745, epoch_train_loss: 0.0604975133312079, epoch: 57
step_train_loss: 0.04281070455908775 train_step: 27500, learning_rate: 4e-05                                  | 254/478 [00:41<00:36,  6.07it/s]
epoch 58 iter 477: train loss 0.08268. lr 4.0000e-05: 100%|                                                   | 478/478 [01:18<00:00,  6.08it/s]
test loss: %f 0.013860434621434521
epoch_valid_loss: 0.013860434621434521, epoch_train_loss: 0.059904210863705086, epoch: 58
step_train_loss: 0.037258751690387726 train_step: 28000, learning_rate: 4e-05                                 | 276/478 [00:45<00:33,  6.11it/s]
epoch 59 iter 477: train loss 0.05775. lr 4.0000e-05: 100%|                                                   | 478/478 [01:19<00:00,  6.05it/s]
test loss: %f 0.012902738625632116
epoch_valid_loss: 0.012902738625632116, epoch_train_loss: 0.058687997685753154, epoch: 59
Saving at epoch 59: ./cond_gpt/weights/None_lengthsplit_50layer_2head_16embd_32bs.pt
step_train_loss: 0.04941820725798607 train_step: 28500, learning_rate: 4e-05                                  | 298/478 [00:49<00:29,  6.01it/s]
epoch 60 iter 477: train loss 0.08247. lr 4.0000e-05: 100%|                                                   | 478/478 [01:19<00:00,  5.99it/s]
test loss: %f 0.013764038897567877
epoch_valid_loss: 0.013764038897567877, epoch_train_loss: 0.05798212577156317, epoch: 60

Results Table:
n_layer    n_head     n_embd     validation_loss      time_per_epoch
50         2          16         1.3816234270731609   97.56907749176025
50         2          16         1.0268689802399389   86.88305497169495
50         2          16         0.8456329692293096   86.88909959793091
50         2          16         0.7738873450844376   87.81722927093506
50         2          16         0.7046002171657704   87.01436400413513
50         2          16         0.6336997482511733   87.13222980499268
50         2          16         0.5708381583293279   87.5953061580658
50         2          16         0.4480134127316652   86.93578743934631
50         2          16         0.388396653312223 95 87.4491195678711
50         2          16         0.30689116374210074  87.3046486377716
50         2          16         0.24817123264074326  87.27850127220154
50         2          16         0.20577994264938212  87.33934092521667
50         2          16         0.16961453151371744  87.21852445602417
50         2          16         0.13510421346183177  87.15836668014526
50         2          16         0.11595657950750103  87.0739336013794
50         2          16         0.1352660658734816   86.48729348182678
50         2          16         0.08133871956831878  86.01002764701843
```

```
Results Table:
n_layer    n_head     n_embd     validation_loss      time_per_epoch
50         2          16         1.3816234270731609   97.56907749176025
50         2          16         1.0268689802399389   86.88305497169495
50         2          16         0.8456329692293096   86.88909959793091
50         2          16         0.7738873450844376   87.81722927093506
50         2          16         0.7046002171657704   87.01436400413513
50         2          16         0.6336997482511733   87.13222980499268
50         2          16         0.5708381583293279   87.5953061580658
50         2          16         0.4480134127316652   86.93578743934631
50         2          16         0.38839665331222395  87.4491195678711
50         2          16         0.30689116374210074  87.3046486377716
50         2          16         0.24817123264074326  87.27850127220154
50         2          16         0.20577994264938212  87.33934092521667
50         2          16         0.16961453151371744  87.21852445602417
50         2          16         0.13510421346183177  87.15836668014526
50         2          16         0.11595657950750103  87.0739336013794
50         2          16         0.1352660658734816   86.48729348182678
50         2          16         0.08133871956831878  86.01002764701843
50         2          16         0.11203097965982226  87.3331651687622
50         2          16         0.06970568711834925  87.21651434898376
50         2          16         0.06352007120019859  87.53285455703735
50         2          16         0.12730575766828325  86.09204387664795
50         2          16         0.06330315358246918  86.7722449302673
50         2          16         0.05981088084755121  83.86673831939697
50         2          16         0.047140366604758635 83.62232065200806
50         2          16         0.04317793250083923  83.40511751174927
50         2          16         0.06270570887459649  86.6171176433563
50         2          16         0.06220773842047762  85.15019965171814
50         2          16         0.042814590554270476 86.72601270675659
50         2          16         0.0350393446645251   85.14984083175659
50         2          16         0.031282561727695994 85.82515001296997
50         2          16         0.03438812578786855  84.0733597278595
50         2          16         0.03687326258255376  85.95911145210266
50         2          16         0.029598432248113333 86.35100626945496
50         2          16         0.028342962644442363 85.82388114929199
50         2          16         0.04783162883379393  86.15980648994446
50         2          16         0.02354517739473118  85.59817576408386
```

```
50         2          16         0.05981088084755121  83.86673831939697
50         2          16         0.047140366604758635 83.62232065200806
50         2          16         0.04317793250083923  83.40511751174927
50         2          16         0.06270570887459649  86.61711764335632
50         2          16         0.06220773842047762  85.15019965171814
50         2          16         0.042814590554270476 86.72601270675659
50         2          16         0.0350393446645251   85.14984083175659
50         2          16         0.031282561727695994 85.82515001296997
50         2          16         0.03438812578786855  84.0733597278595
50         2          16         0.03687326258255376  85.95911145210266
50         2          16         0.029598432248113333 86.35100626945496
50         2          16         0.028342962644442363 85.82388114929199
50         2          16         0.04783162883379393  86.15980648994446
50         2          16         0.02354517739473118  85.59817576408386
50         2          16         0.03534271942313622  82.25590395927429
50         2          16         0.02087155810591799  82.59697151184082
50         2          16         0.021083436544156738 83.15034341812134
50         2          16         0.018854224610280385 85.5999858379364
50         2          16         0.01730320260308131  85.44301080703735
50         2          16         0.018228005781700765 86.89180874824524
50         2          16         0.017130475913829827 85.25211119651794
50         2          16         0.016327320253131567 85.5942542552948
50         2          16         0.0159001647552941   87.77797293663025
50         2          16         0.016065317162967944 91.29987859725952
50         2          16         0.015067671697276333 86.02754735946655
50         2          16         0.015491769450751168 85.80844378471375
50         2          16         0.017191762432318042 85.83576774597168
50         2          16         0.015099257135901737 86.19236254692078
50         2          16         0.013171155885275867 85.80043983459473
50         2          16         0.018089427785189065 82.20216989517212
50         2          16         0.013713264096252344 82.27552366256714
50         2          16         0.015097079953799645 82.61570501327515
50         2          16         0.014583172395187258 81.53680920600891
50         2          16         0.012903952738270164 82.076913356781
50         2          16         0.014689087841866745 81.84507703781128
50         2          16         0.013860434621434521 81.7358467578888
50         2          16         0.012902738625632116 82.06283044815063
50         2          16         0.013764038897567877 82.78730416297913
```

```
[apurva.mandalika@g009 code]$ python main.py --task generate --data_split 'length' --n_layer 50
The file './tokenizer/length_vocab.json' exists. Loading tokenizer.
{'<pad>': 0, '<s>': 1, '</s>': 2, '<unk>': 3, 'I_WALK': 4, 'I_JUMP': 5, 'I_LOOK': 6, 'I_TURN_RIGHT': 7, 'I_RUN': 8, 'I_TURN_LEFT': 9, 'walk': 10, 'jump': 11, 'look': 12, 'turn': 13,
'right': 14, 'run': 15, 'left': 16, 'after': 17, 'twice': 18, 'opposite': 19, 'and': 20, 'thrice': 21, 'around': 22}
loading model
total params: 166848
Accuracy: 0.0000: 100%|                                                                                                                                  | 3920/3920 [4:01:56<00:00,  3.70s/it]
Test accuracy: 0.0
[apurva.mandalika@g009 code]$
```