1) You are asked to run 12 experiments with different architectures, optimizers, and hyperparameters settings. These parameter settings are given to you at the top of the runner file (run exp.py). For each of these 23 experiments, plot learning curves (train and validation) of perplexity over both epochs and wall-clock-time. Figures should have labeled axes and a legend and an explanatory caption.

- Experiment 1:

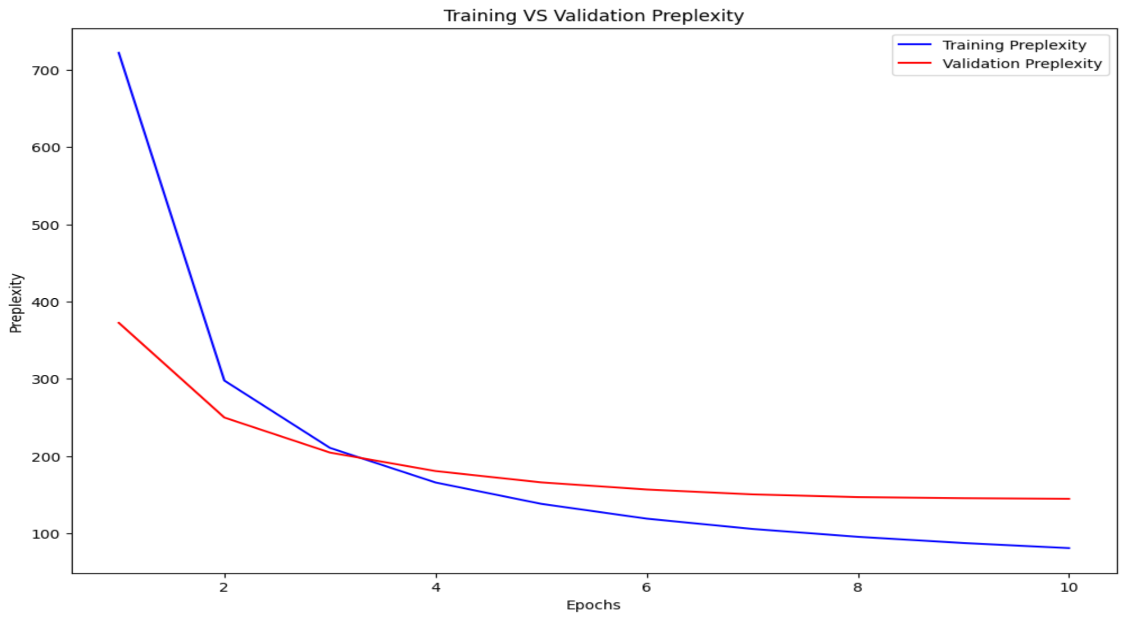

*Table 1: model='lstm', layers=1, batch_size=16, log=True, epochs=10, optimizer='adam'*

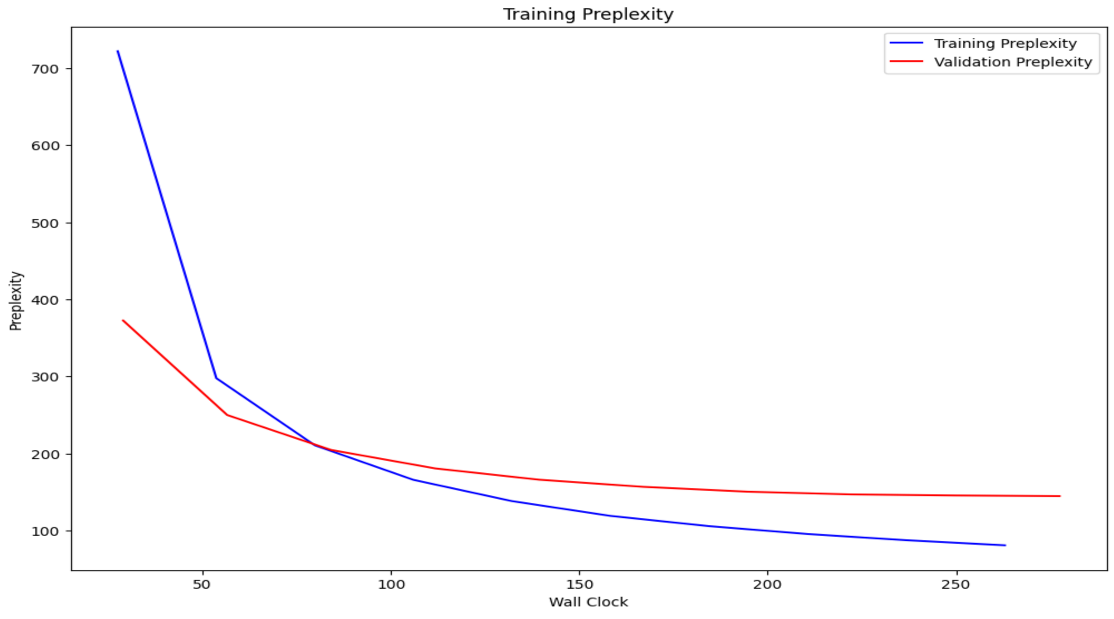

*Table 2:model='lstm', layers=1, batch_size=16, log=True, epochs=10, optimizer='adam'*

## ❖ Experiment 2:

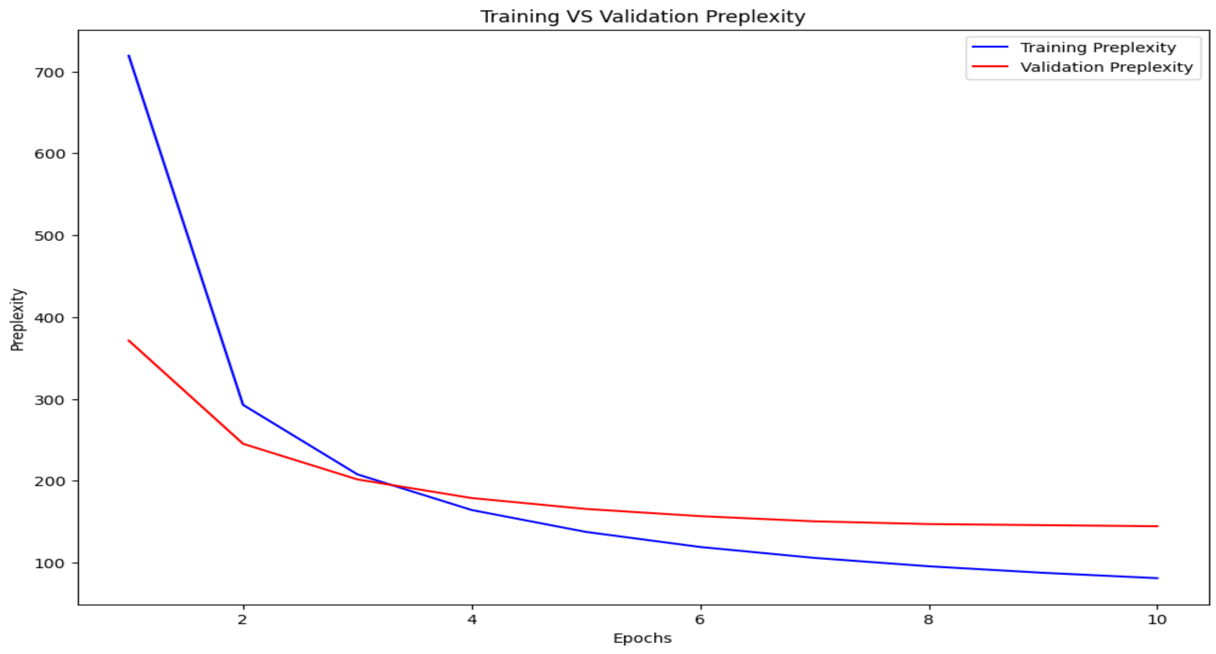

*Table 3: model='lstm', layers=1, batch_size=16, log=True, epochs=10, optimizer='adamw'*

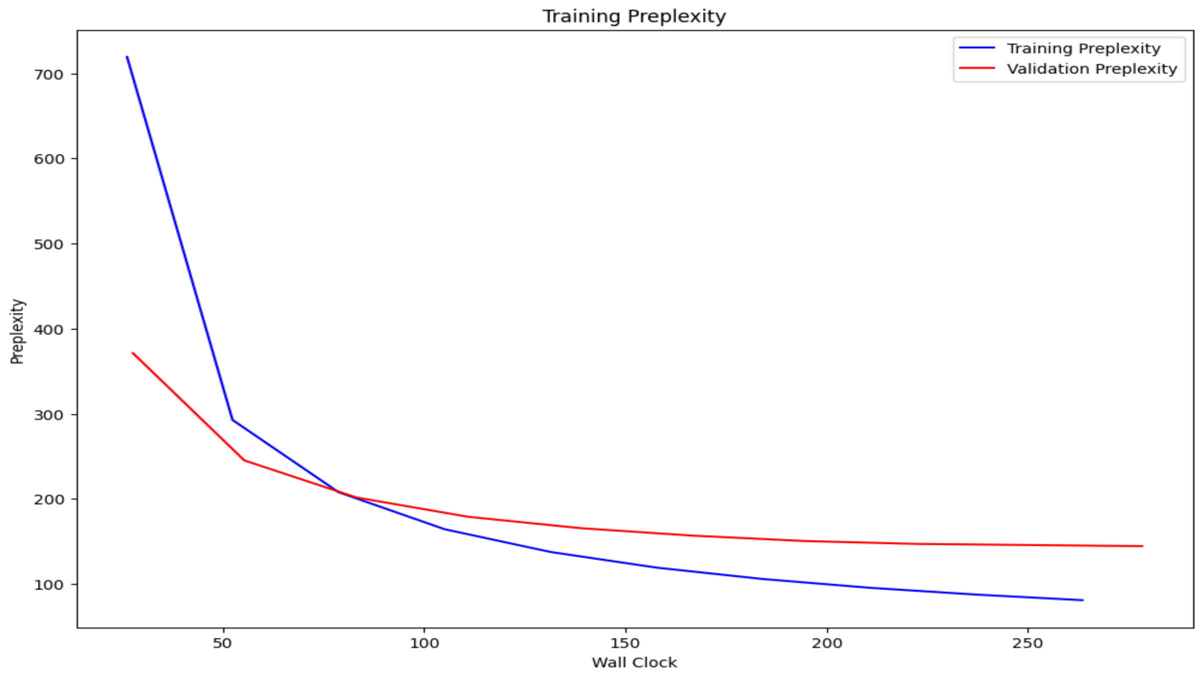

*Table 4: model='lstm', layers=1, batch_size=16, log=True, epochs=10, optimizer='adamw'*

## ❖ Experiments 3:

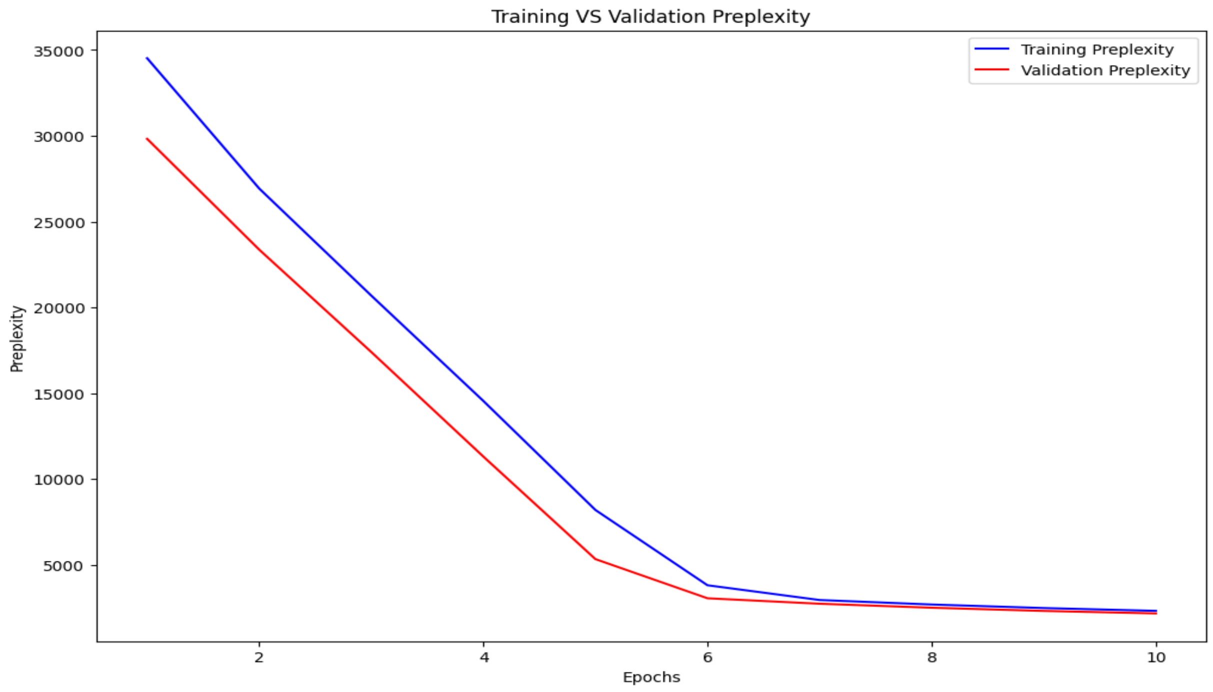

*Table 5: model='lstm', layers=1, batch_size=16, log=True, epochs=10, optimizer='sgd'*

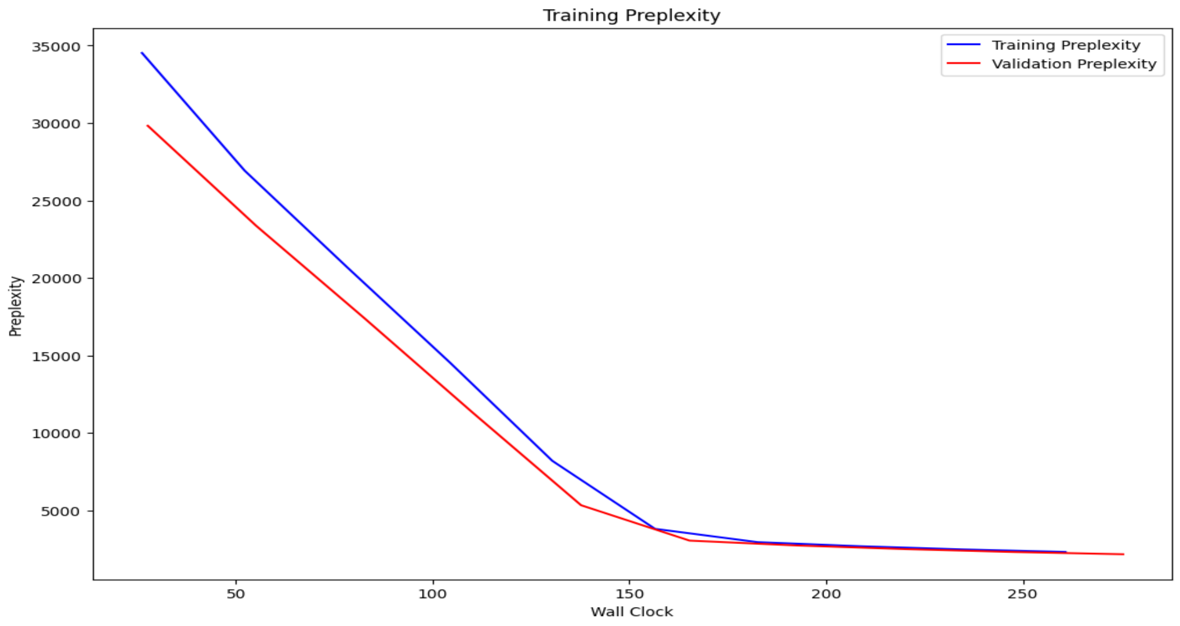

*Table 6:model='lstm', layers=1, batch_size=16, log=True, epochs=10, optimizer='sgd'*

## ❖ Experiment 4:

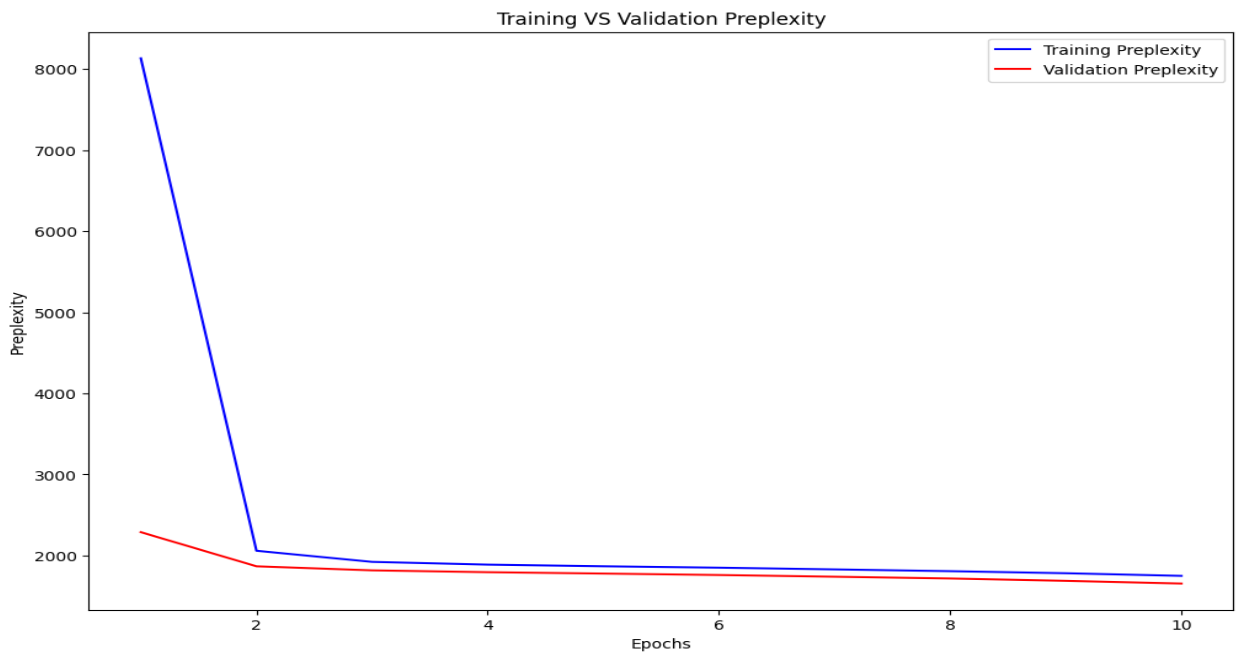

*Table 7:model='lstm', layers=1, batch_size=16, log=True, epochs=10, optimizer='momentum'*

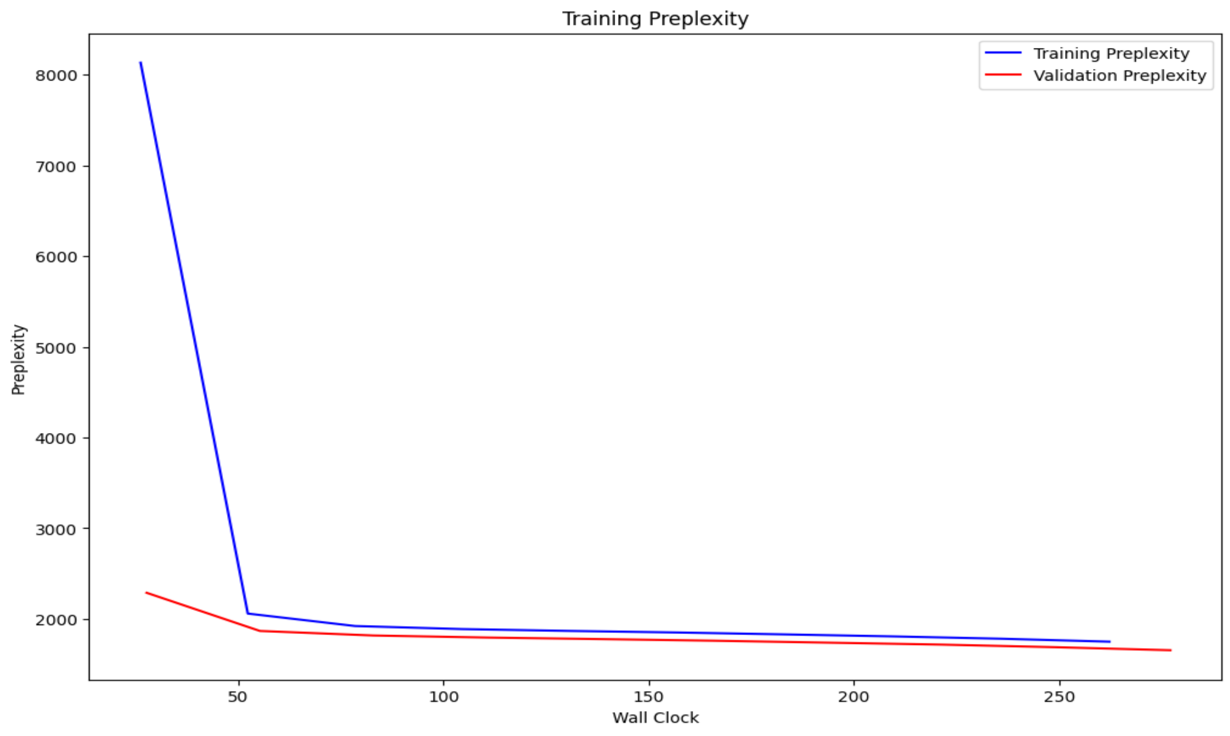

*Table 8:model='lstm', layers=1, batch_size=16, log=True, epochs=10, optimizer='momentum'*

## ❖ Experiment 5:

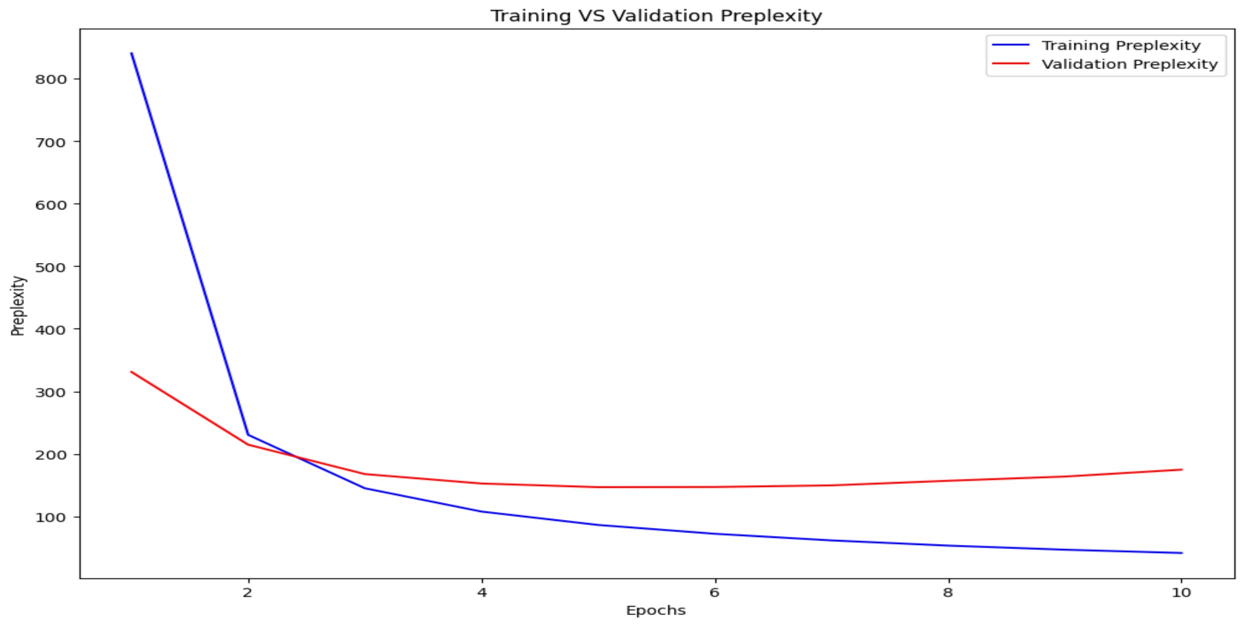

*Table 9:model='gpt1', layers=1, batch_size=16, log=True, epochs=10, optimizer='adam'*

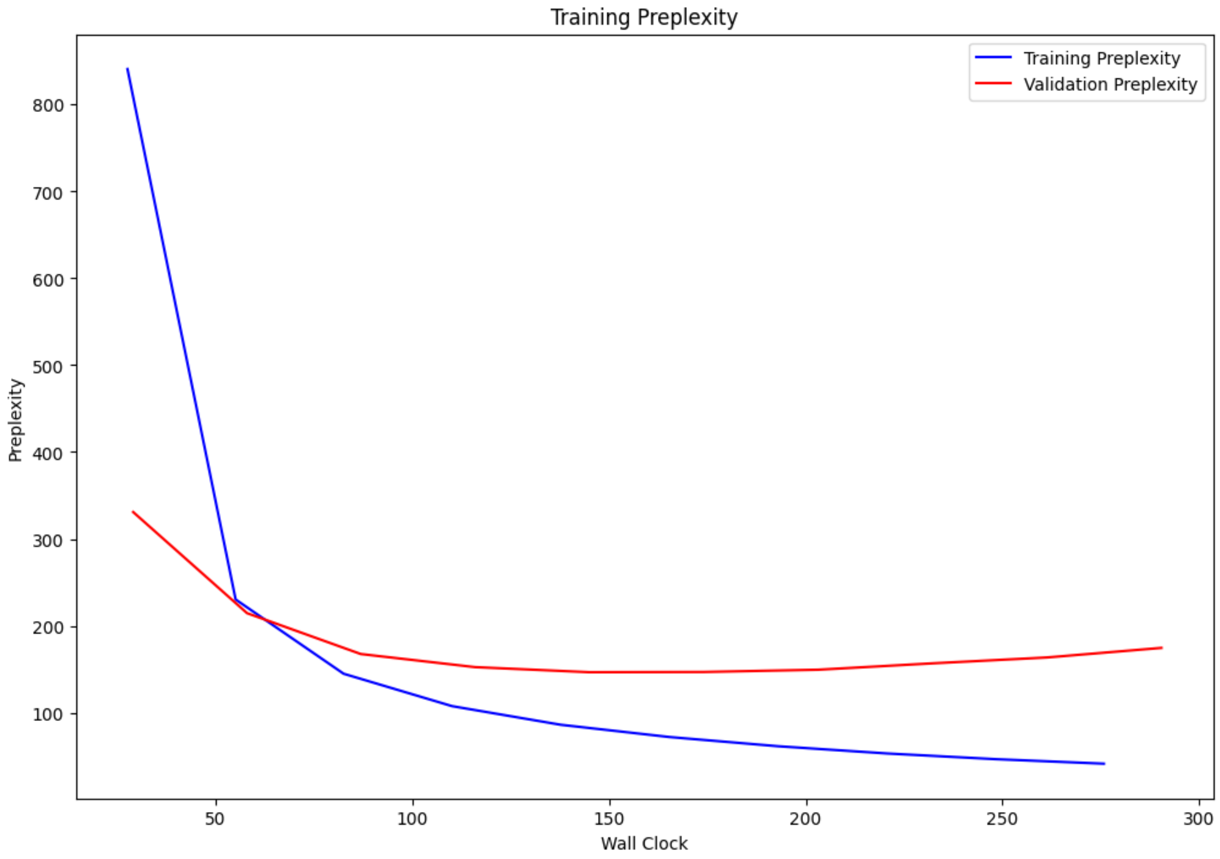

*Table 10: model='gpt1', layers=1, batch_size=16, log=True, epochs=10, optimizer='adam'*

## ❖ Experiment 6:

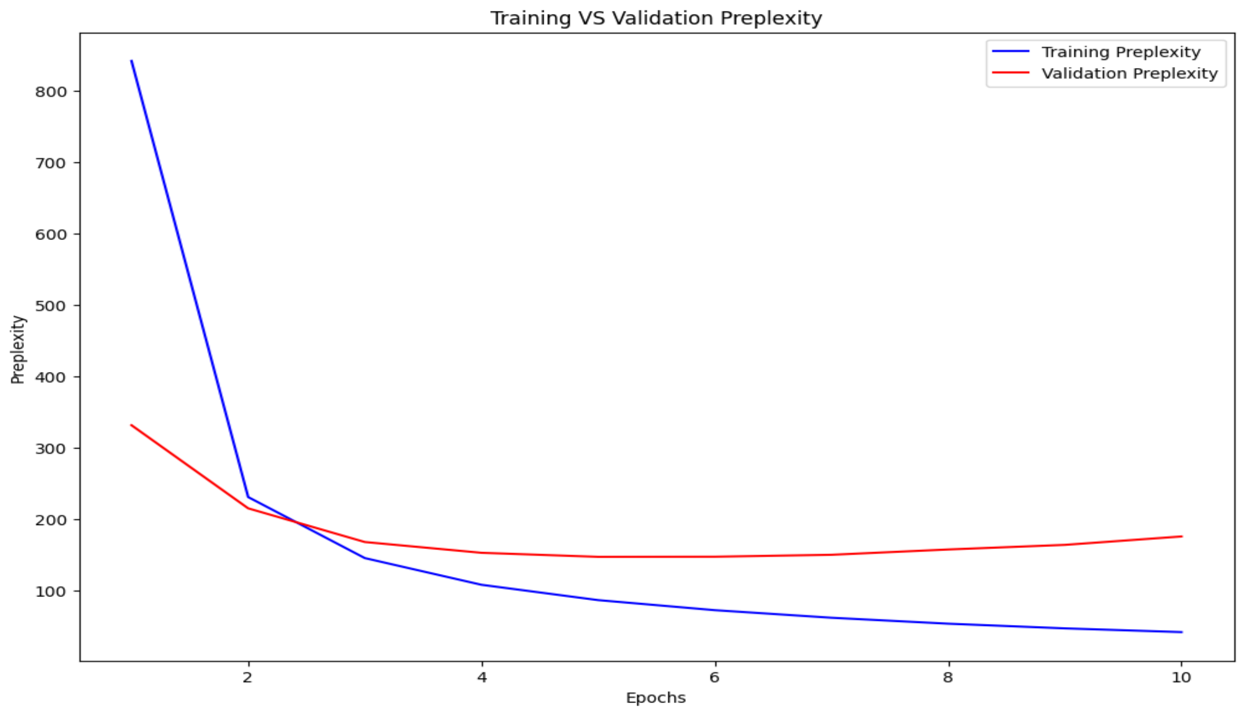

*Table 11:model='gpt1', layers=1, batch_size=16, log=True, epochs=10, optimizer='adamw'*

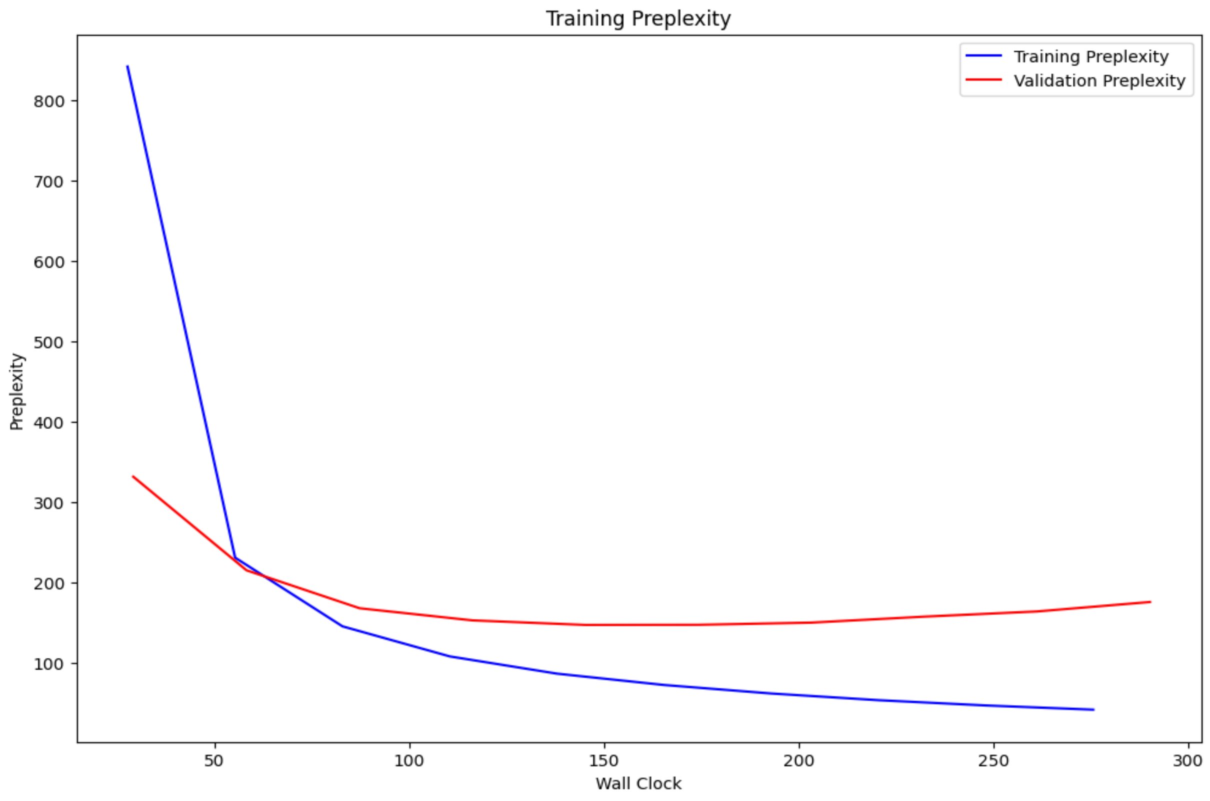

*Table 12: model='gpt1', layers=1, batch_size=16, log=True, epochs=10, optimizer='adamw'*

## ❖ Experiment 7:

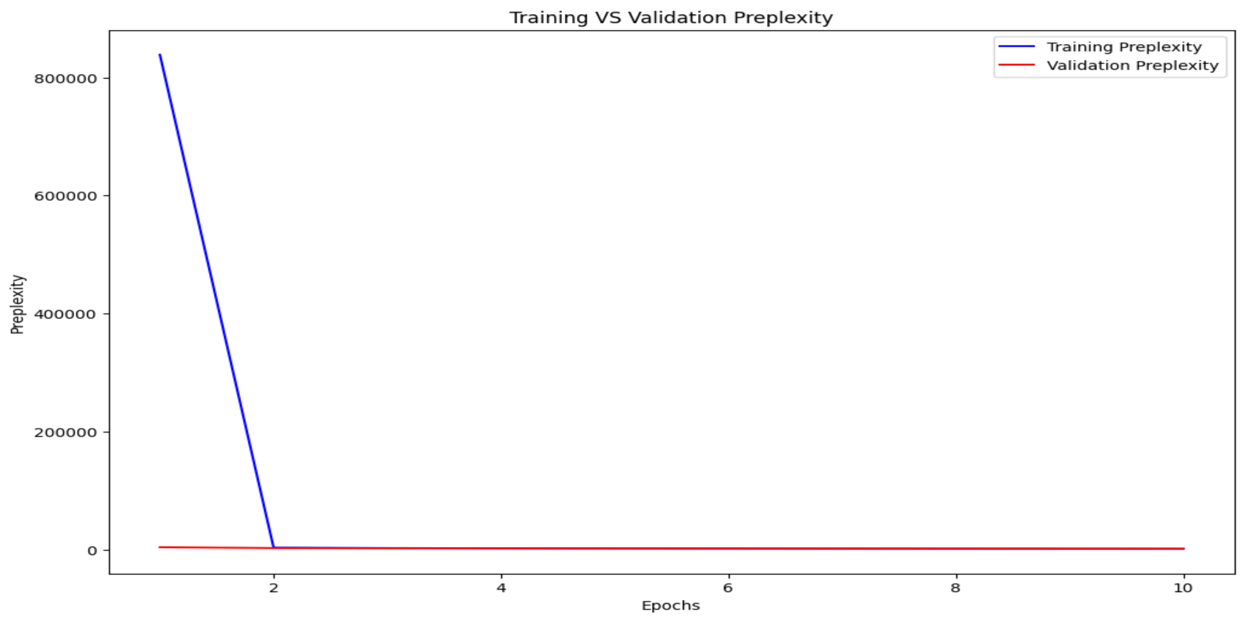

*Table 13: model='gpt1', layers=1, batch_size=16, log=True, epochs=10, optimizer='sgd'*

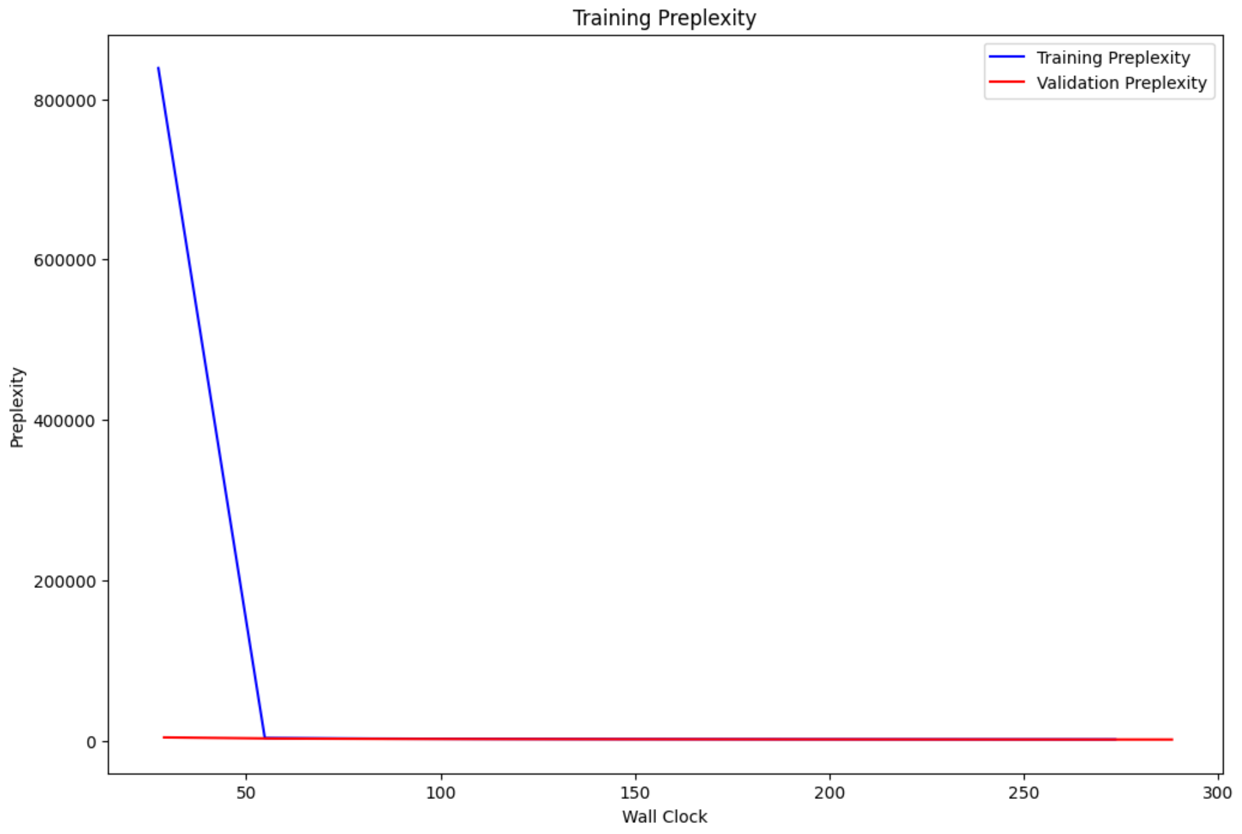

*Table 14: model='gpt1', layers=1, batch_size=16, log=True, epochs=10, optimizer='sgd'*

## ❖ Experiment 8:

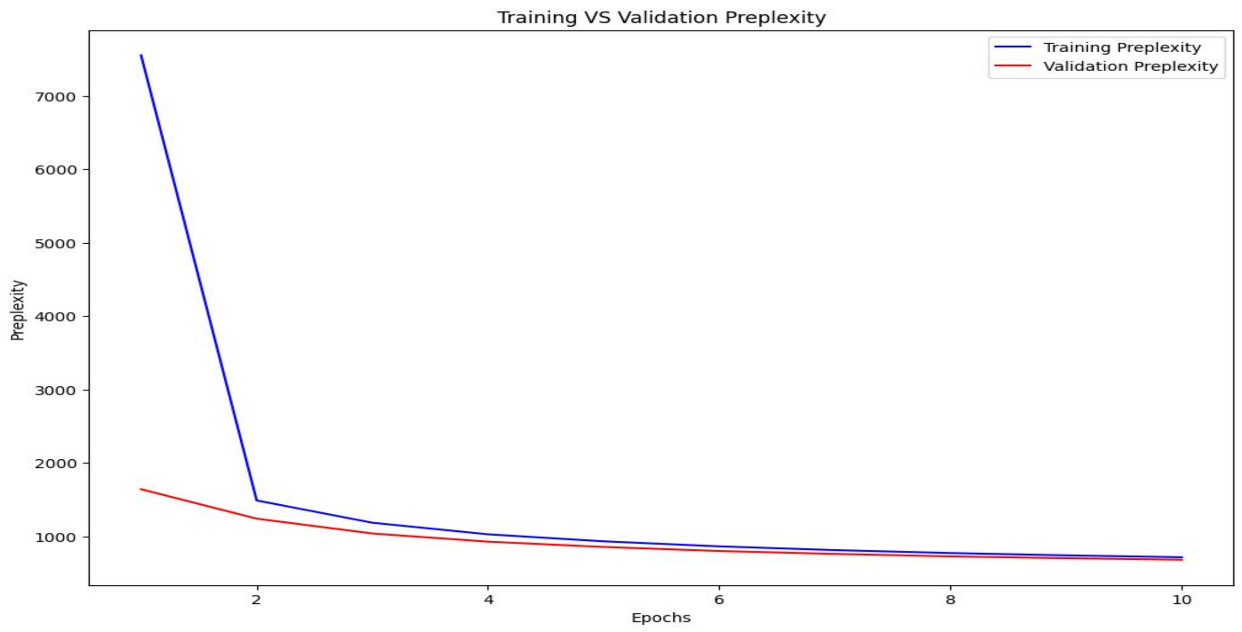

*Table 15:model='gpt1', layers=1, batch_size=16, log=True, epochs=10, optimizer='momentum'*

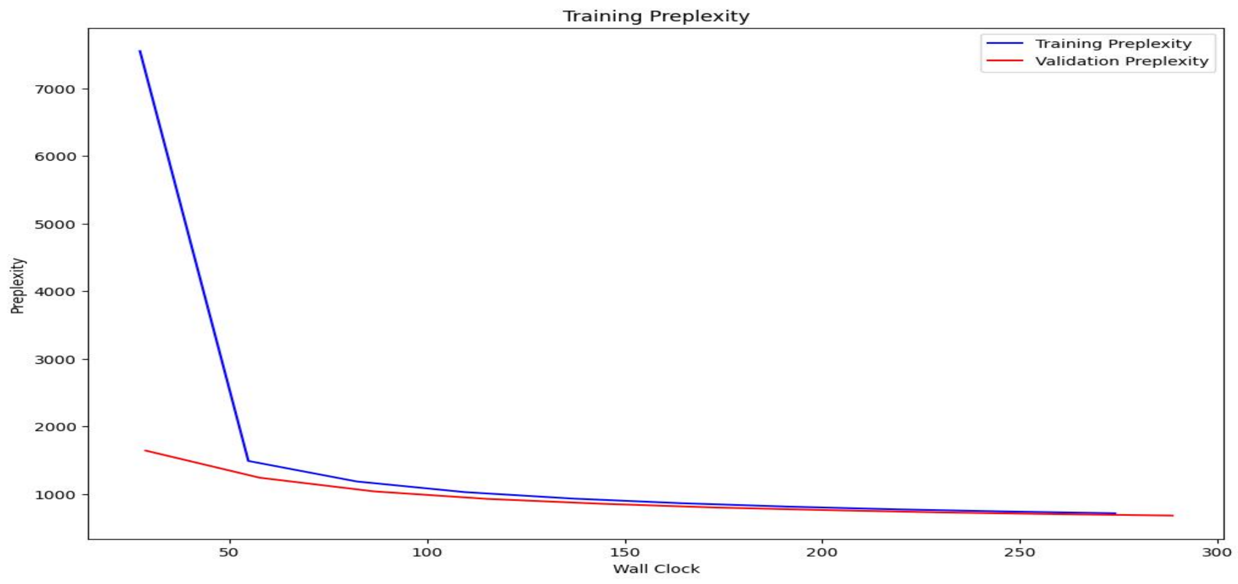

*Table 16:model='gpt1', layers=1, batch_size=16, log=True, epochs=10, optimizer='momentum'*

## ❖ Experiment 9:

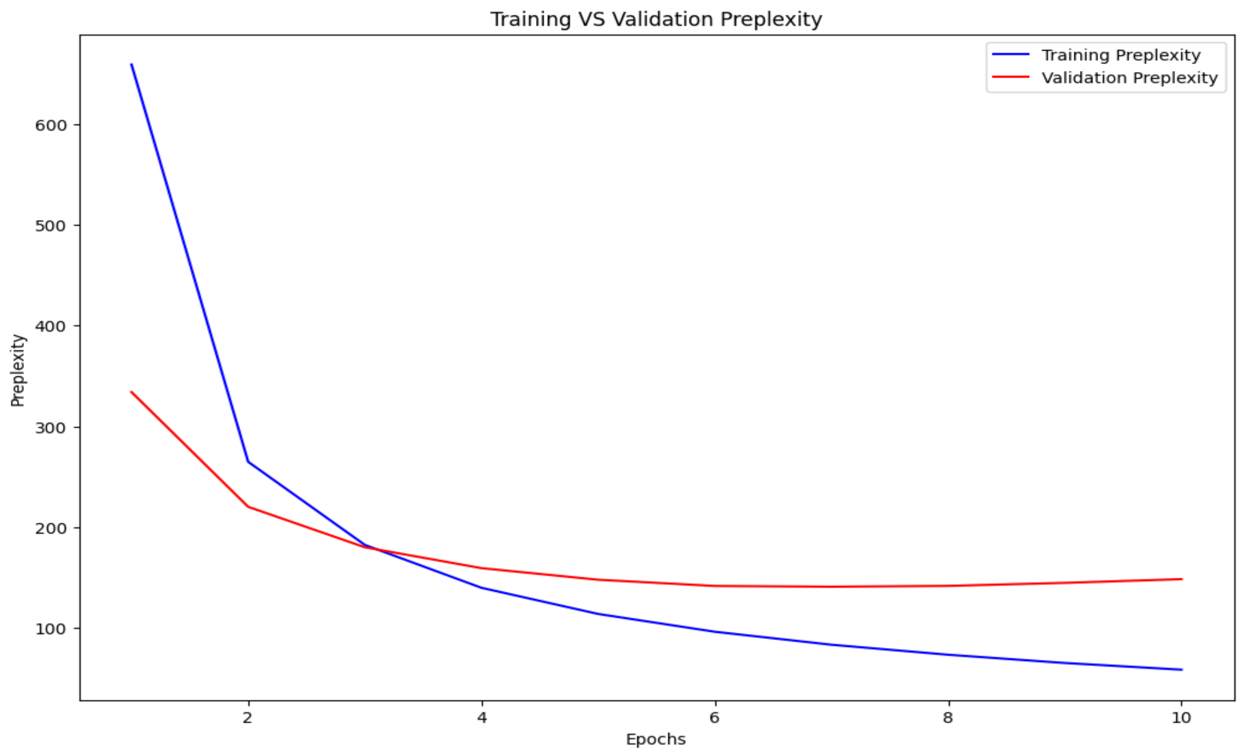

*Table 17:model='lstm', layers=2, batch_size=16, log=True, epochs=10, optimizer='adamw'*

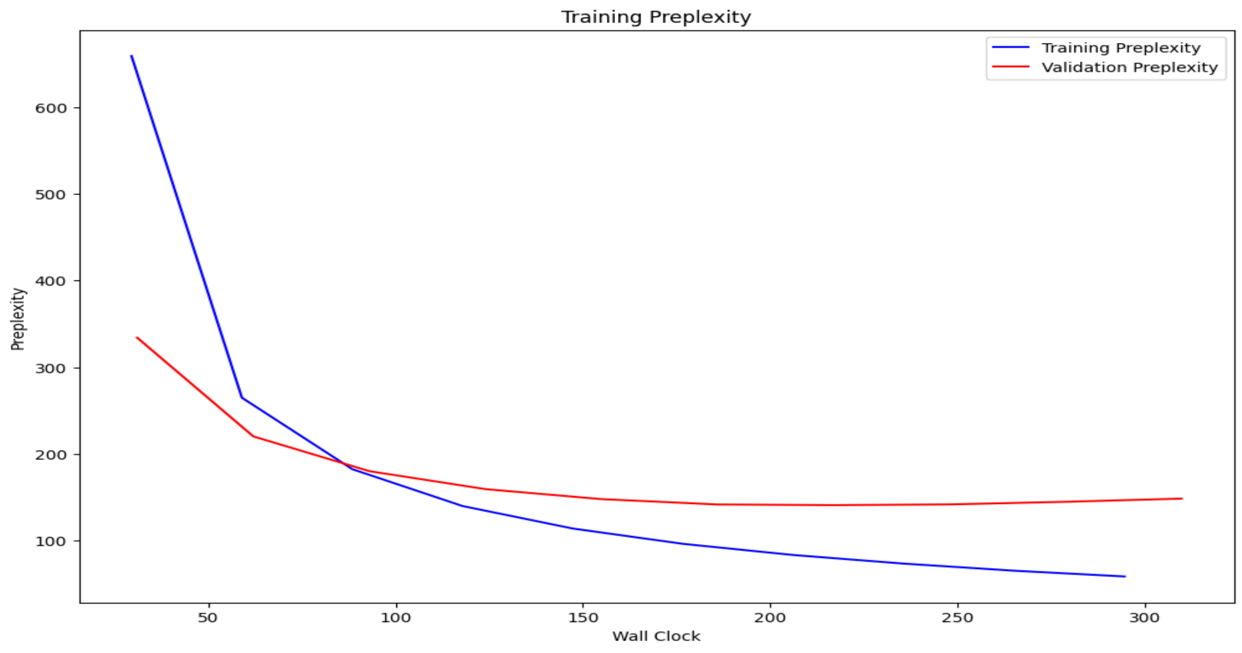

*Table 18:model='lstm', layers=2, batch_size=16, log=True, epochs=10, optimizer='adamw'*

## ❖ Experiment 10:

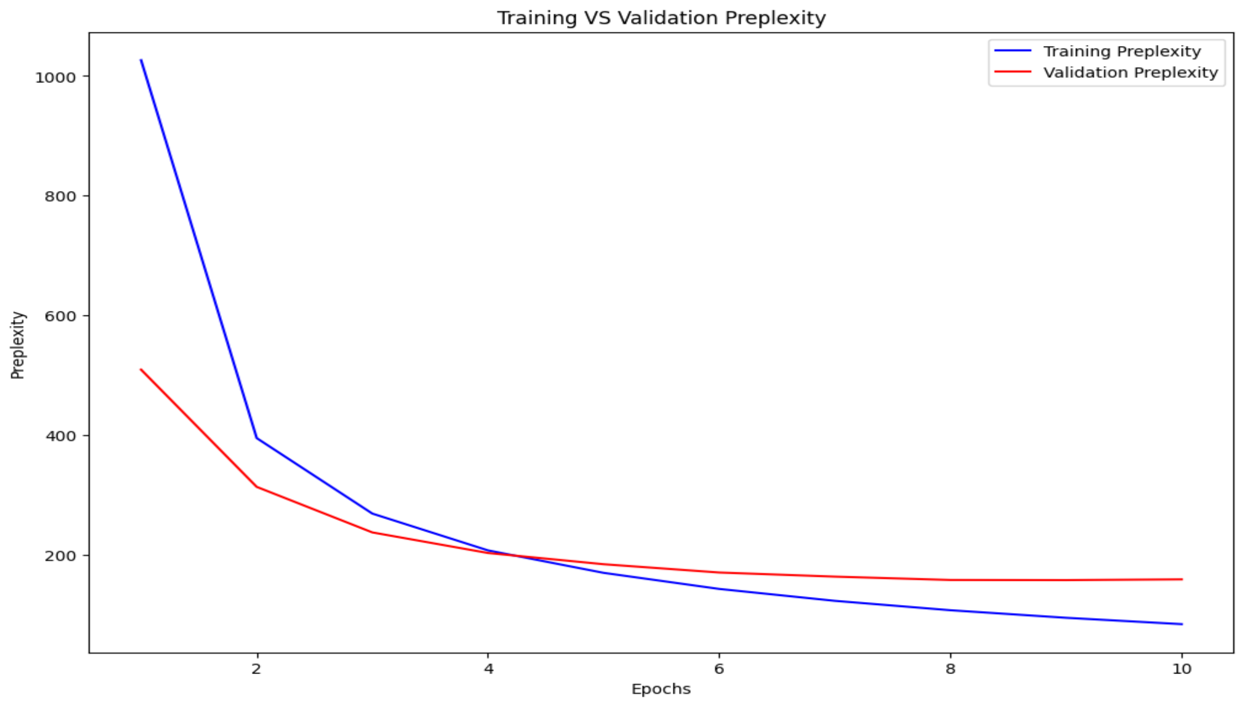

*Table 19: model='lstm', layers=4, batch_size=16, log=True, epochs=10, optimizer='adamw'*

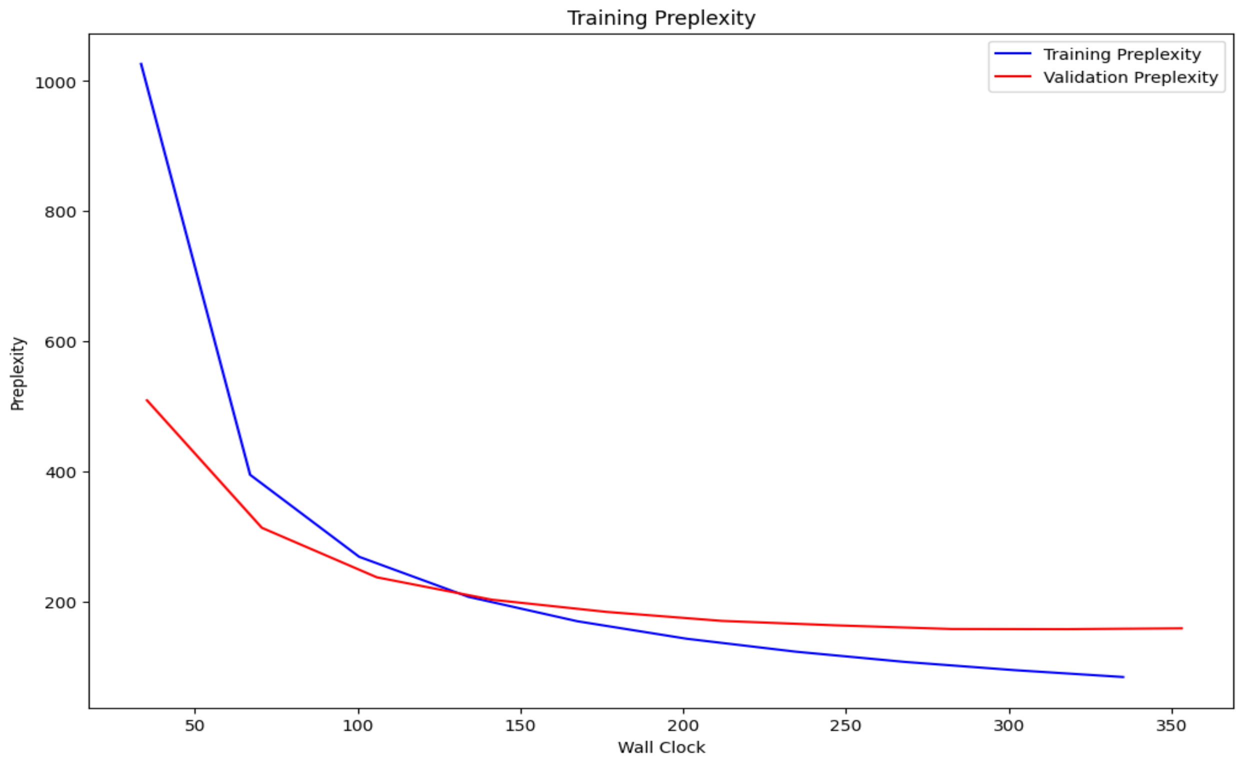

*Table 20:model='lstm', layers=4, batch_size=16, log=True, epochs=10, optimizer='adamw'*

## ❖ Experiment 11:

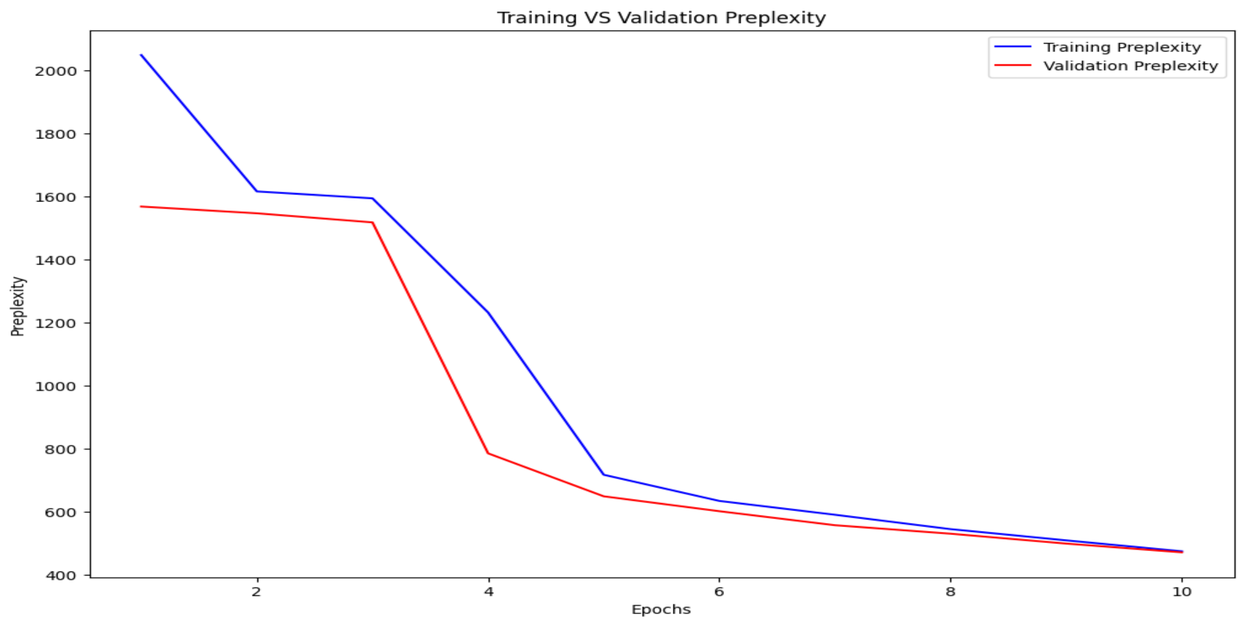

*Table 21:model='gpt1', layers=2, batch_size=16, log=True, epochs=10, optimizer='adamw'*

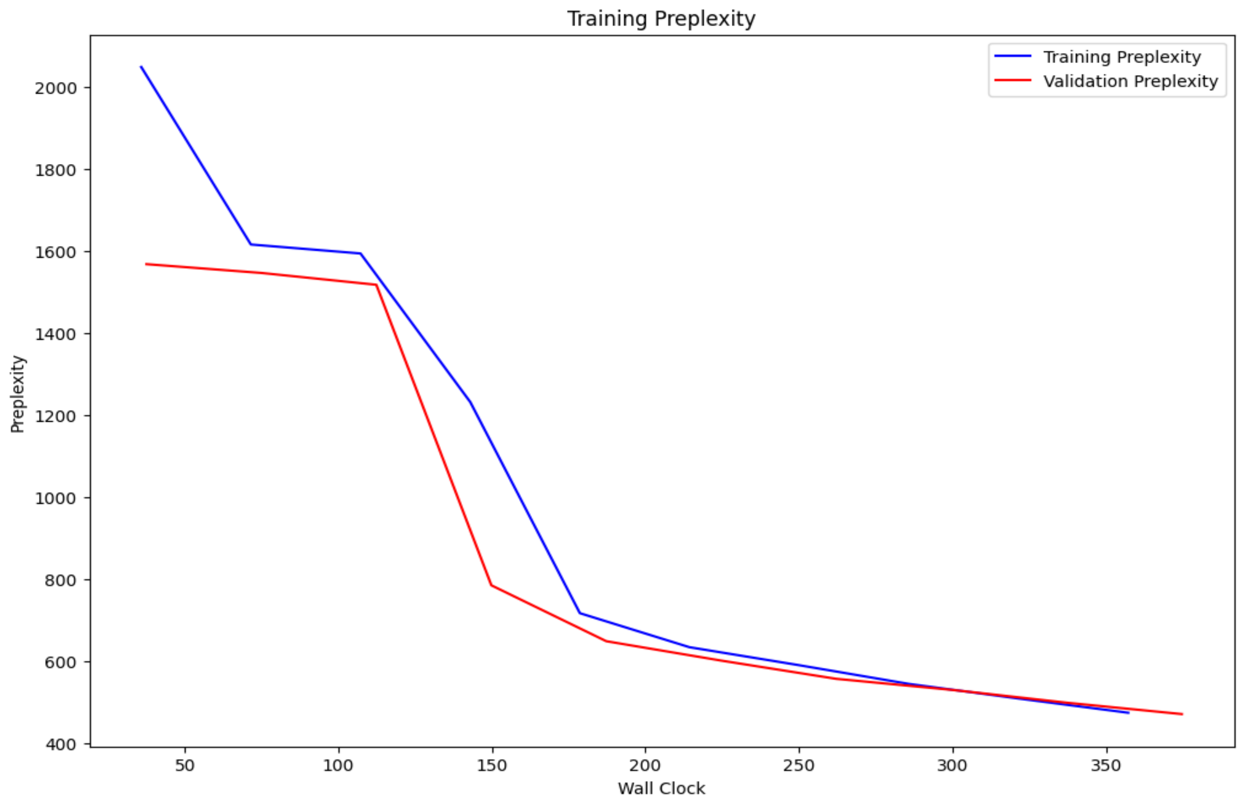

*Table 22:model='gpt1', layers=2, batch_size=16, log=True, epochs=10, optimizer='adamw'*

## ❖ Experiment 12:

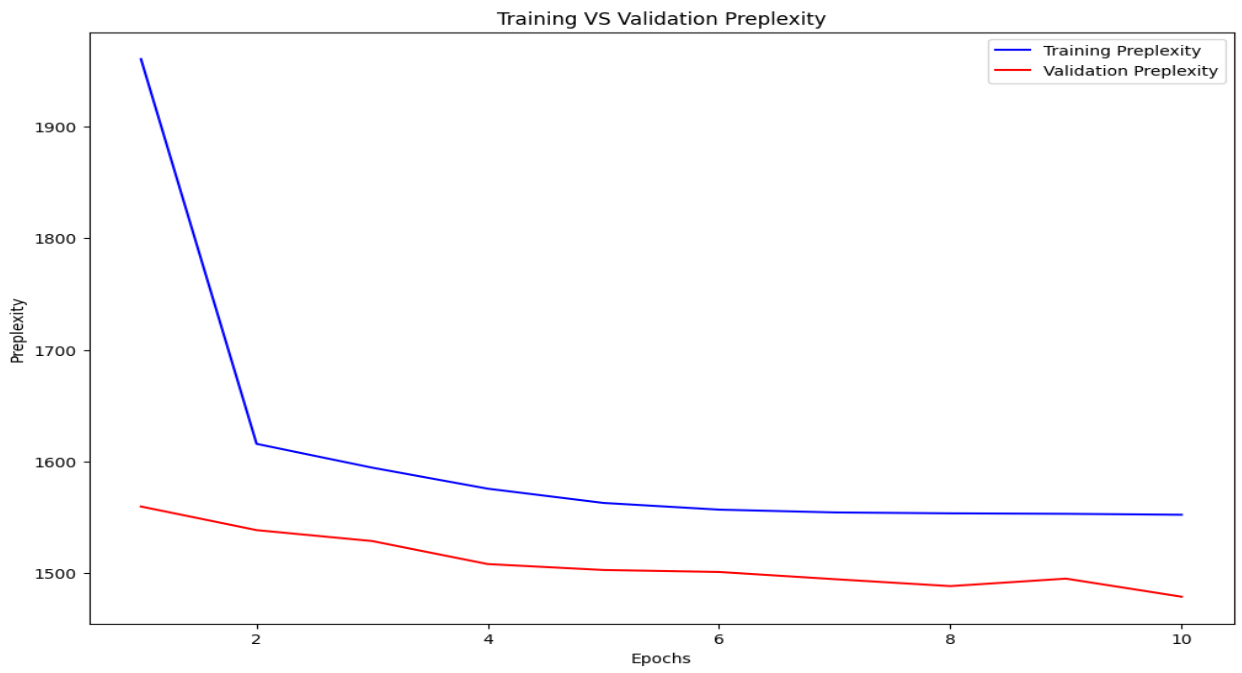

*Table 23:model='gpt1', layers=4, batch_size=16, log=True, epochs=10, optimizer='adamw'*

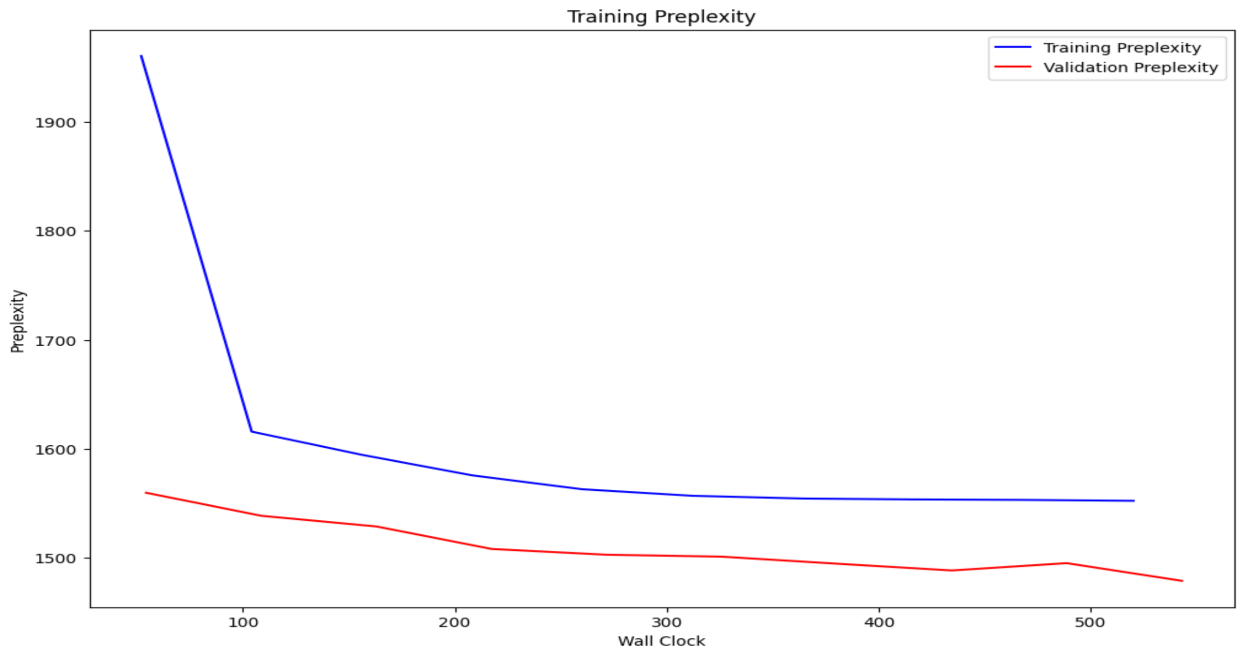

*Table 24:model='gpt1', layers=4, batch_size=16, log=True, epochs=10, optimizer='adamw'*

2) Make a table of results summarizing the train and validation performance for each experiment, indicating the architecture and optimizer. Sort by architecture, then number of layers, then optimizer, and use the same experiment numbers as in the runner script for easy reference. Bold the best result for each architecture.2 The table should have an explanatory caption, and appropriate column and/or row headers. Any shorthand or symbols in the table should be explained in the caption.

*Table 25: Comparative Performance Metrics of LSTM and GPT-1 Models Across Various Hyperparameter Configurations: This table showcases the results of experiments with different numbers of layers and optimizers, highlighting training, validation, and test perplexities (Preplex), losses, model parameter count, learnable parameters, and GPU memory usage, to analyze the impact of model architecture and complexity on learning efficiency and potential overfitting tendencies.*

| #Expe | Model | Optimizer | #Layer | Train Preplex[1] | Vali Preplex[2] | Test Preplex[3] | Train Loss | Vali Loss[4] | Test Loss | Model Param[5] | Learnable Param[6] | GPU |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | lstm | adam | 1 | 81.06 | 144.84 | 148.86 | 4.39 | 4.97 | 5.00 | 34.1M | 30M | 4.7GB |
| **2** | **lstm** | **adamw** | **1** | **81.07** | **144.50** | **149.58** | **4.39** | **4.97** | **5.00** | **34.1M** | **30M** | **4.7GB** |
| 3 | lstm | momentum | 1 | 1748.33 | 1654.09 | 1600.03 | 7.46 | 7.41 | 7.37 | 34.1M | 30M | 4.7GB |
| 4 | lstm | sgd | 1 | 2322.22 | 2173.39 | 2088.85 | 7.75 | 7.68 | 7.64 | 34.1M | 30M | 3.8GB |
| 5 | lstm | adamw | 2 | 58.65 | 148.45 | 153.10 | 4.07 | 5.00 | 5.03 | 36.2M | 5.1M | 4.7GB |
| 6 | lstm | adamw | 4 | 84.80 | 159.57 | 164.58 | 4.440 | 5.072 | 5.10 | 40.4M | 9.3M | 4.8GB |
| 7 | gpt1 | adam | 1 | 41.24 | 174.55 | 178.21 | 3.71 | 5.16 | 5.18 | 38.4M | 7.1M | 3.5GB |
| **8** | **gpt1** | **adamw** | **1** | **41.20** | **174.12** | **178.76** | **3.71** | **5.16** | **5.18** | **38.4M** | **7.1M** | **3.51GB** |
| 9 | gpt1 | momentum | 1 | 714.16 | 679.82 | 666.62 | 6.57 | 6.52 | 6.50 | 38.4M | 7.1M | 3.51GB |
| 10 | gpt1 | sgd | 1 | 1617.57 | 1500.89 | 1476.06 | 7.38 | 7.31 | 7.29 | 38.4M | 7.1M | 3.51GB |
| 11 | gpt1 | adamw | 2 | 473.19 | 469.95 | 449.53 | 6.15 | 6.15 | 6.10 | 45.5M | 14.2M | 3.8GB |
| 12 | gpt1 | adamw | 4 | 1552.26 | 1478.83 | 1442.59 | 7.34 | 7.29 | 7.27 | 59.6M | 28.4M | 4.45GB |

3) Which hyperparameters + optimizer would you use if you were most concerned with wall-clock time? With generalization performance?

When prioritizing training duration, the aim is to identify a setup that minimizes training time while maintaining acceptable model performance. From the results of my experiments, it appears that the most efficient configuration in terms of speed is the single-layer GPT-1 using either the Adamw optimizer. This is supported by evidence from my logs and charts, which show this setup achieving a perplexity of around 330 in approximately 50 seconds. When it comes to generalization that is, how well a model predicts on new, unseen data, the one-layer LSTM model with either AdamW optimizers emerges as the most effective. This conclusion is drawn from the smaller disparity between validation and test perplexity in my experimental outcomes.

4) Between the experiment configurations 1-4 and 5-8 at the top of run exp.py, only the optimizer changed. What difference did you notice about the four optimizers used? What was the impact of weight decay, momentum, and Adam?

In the experiments numbered 1 and 7, the Adam optimizer was employed, notable for its adaptive learning rate capability, which typically allows for quicker convergence in comparison to the classic stochastic gradient descent (SGD) method. The LSTM model in Experiment 1 exhibited moderate success in terms of training and validation perplexity, but it didn't top the charts among the tested optimizers. Conversely, the GPT-1 model in Experiment 7 displayed a commendable low training perplexity, hinting at effective learning; however, its higher validation

[1] Train Perplexity
[2] Validation Perplexity
[3] Test Perplexity
[4] Validation Loss
[5] Model Parameter
[6] Learnable Parameter

perplexity raises concerns about possible overfitting. Turning to Experiments 2 and 8, the AdamW optimizer, which integrates weight decay as a regularizing feature to combat overfitting, was analyzed. The LSTM model's performance in Experiment 2 mirrored the results obtained with Adam, indicating that the addition of weight decay had little to no effect on its performance. On the other hand, for the GPT-1 model in Experiment 8, there was a marginal improvement in validation perplexity with AdamW compared to Adam, suggesting a beneficial role for weight decay in enhancing the model's generalization capabilities.

For Experiments 3 and 9, leveraging the momentum optimizer, which theoretically should hasten convergence by guiding gradient updates more effectively, yielded mixed results. In the case of the LSTM model in Experiment 3, the notably higher perplexity implies that momentum may not be suited for this configuration or might need more precise tuning. However, for the GPT-1 model in Experiment 9, while momentum did manage to lower the validation perplexity compared to Adam, it still fell short of the performance achieved with AdamW, underscoring the latter's superior contribution to generalization. Lastly, in Experiments 4 and 10, the SGD optimizer was assessed. Known for its simplicity and heavy reliance on learning rate settings, SGD required additional epochs to converge. It proved to be the least efficient optimizer in this suite of experiments, as demonstrated by the LSTM model's highest perplexity in Experiment 4. This trend continued with the GPT-1 model in Experiment 10, where the performance lagged behind that of the other optimizers, further solidifying the conclusion that SGD was the least effective optimizer in these tests. Overall, Adam and AdamW emerged as the superior choices for both the LSTM and GPT-1 models, with AdamW potentially providing a slight edge in generalization due to its weight decay component. Momentum did not surpass the performance of the Adam variants and only modestly outdid SGD.

5) Compare experiments 1 and 5. Which model did you think performed better (LSTM or GPT)? Why?

In the first experiment where the LSTM model was paired with the Adam optimizer, the outcomes were satisfactory during training as seen in the perplexity and loss metrics. However, the model displayed elevated perplexity scores in validation and testing phases, hinting at a tendency to overfit to the training data. The seventh experiment, utilizing the GPT-1 architecture with the same Adam optimizer, demonstrated a stronger grasp of the training data, evidenced by its lower training perplexity in comparison to the LSTM model. Yet, this model's validation perplexity was higher, raising concerns about its ability to generalize as effectively. When assessing the two models with a focus on generalization judged by validation and test metrics the LSTM model may be deemed to have a superior performance, given its comparatively lower perplexity in these areas. It's worth noting that transformer-based models like GPT-1 have the potential to outperform LSTM models, but the efficacy of these models is contingent on them being trained on sufficiently large datasets.

6) In configurations 5-8 and in configurations 11 and 12, you trained a transformer with various hyper-parameter settings. Given the recent high profile transformer based language models, are the results as you expected? Speculate as to why or why not.

In experiments 5-8, 11 and 12, the GPT-1 model's performance was expected to demonstrate the strengths of transformer technology, known for efficient learning and strong generalization. However, any deviation from these expectations could be attributed to a range of factors. A less than optimal learning rate might indicate a need for more training data, better hyperparameter tuning, or a more complex model architecture. Generalization issues could point to an overfitting problem, potentially exacerbated by a lack of regularization techniques. The smaller scale of GPT-1 compared to larger transformer models could also limit its performance, particularly if the dataset size is modest. Furthermore, adding more layers to the model can increase its learning capacity but also raise the risk of overfitting if the data is insufficient. As I mentioned before (Q3.5), the success of the GPT-1 experiments would validate the transformer architecture's effectiveness at a smaller scale, assuming proper training and adequate data representation. Conversely, any shortcomings would likely be due to limitations in model complexity, data availability.

7) For each of the experiment configurations above, measure the average steady-state GPU memory usage (nvidia-smi is your friend!). Comment about the GPU memory footprints of each model, discussing reasons behind increased or decreased memory consumption where applicable.

In my experiments, LSTM models (Experiments 1, 2, 3, 5) maintained a consistent GPU memory usage of 4.7GB across different numbers of layers and optimizers, while a two-layer LSTM with SGD (Experiment 4) unexpectedly used less memory at 3.8GB. A four-layer LSTM (Experiment 6) slightly increased memory use to 4.8GB. Single-layer GPT-1 models (Experiments 7-10) showed efficient memory usage between 3.5GB and 3.51GB, unaffected by the type of optimizer. Memory demands rose modestly to 3.8GB for a two-layer GPT-1 (Experiment 11) and peaked at 4.45GB for a four-layer GPT-1 (Experiment 12), following the expected pattern of increased memory with more layers. Overall, GPT-1 models demonstrated a predictable increase in memory with additional layers, whereas LSTM memory usage remained relatively stable, and the lower memory footprint for GPT-1 as opposed to LSTMs suggests efficiency in handling memory-intensive operations like backpropagation through time.

8) Comment on the overfitting behavior of the various models you trained, under different hyperparameter settings. Did a particular class of models overfit more easily than the others? Can you make an informed guess of the various steps a practitioner can take to prevent overfitting in this case? (You might want to refer to sets of experiments 2, 9, 10 for the LSTM and 6, 11, 12 for GPT – that evaluate models of increasing capacities).

In the LSTM-related experiments (2, 9, 10), an increase in the number of layers did not necessarily heighten the risk of overfitting, as the models did not display a pronounced ability to absorb finer details of the training data. The performance metrics across training, validation, and testing for the 1-layer, 2-layer, and 4-layer LSTMs (Experiments 2, 5, and 6) showed minimal discrepancies, indicating a stable generalization capability across these configurations. Conversely, in the GPT-related experiments (6, 11, 12), the inherent complexity of transformer-based models with their substantial parameter count introduces a heightened risk of overfitting, particularly when training on datasets lacking size and variety. This tendency is evident in the GPT experiments where increasing the number of layers from one to two and then four resulted in validation perplexities of 174.55, 469.95, and 1478.83, respectively, illustrating a progressive overfitting as the models became more complex. Such a trend underscores that transformers, unlike LSTMs in these experiments, become increasingly sensitive to overfitting with the addition of layers, requiring careful handling of model capacity to ensure robust performance. To mitigate overfitting, practitioners are advised to enrich their dataset through data augmentation, providing a wider array of examples for the model to train on, which promotes the learning of more generalized patterns. Additionally, streamlining the model by minimizing the number of layers or parameters can prevent the model from becoming overly complex relative to the size and diversity of the data. Lastly, employing regularization methods, such as L2 regularization or weight decay, can inhibit the model from aligning too closely with the training data, thereby fostering better generalization.