Exploring Multiprocessing through numba, jit, cuda in Python and an Ablation studies over a Machine Translation model using Transformers Architecture

Team 9

Ketaki Kolhatkar
Luv Verma

Motivation: Part 1

Approximate GPU RAM needed to store 1B parameters

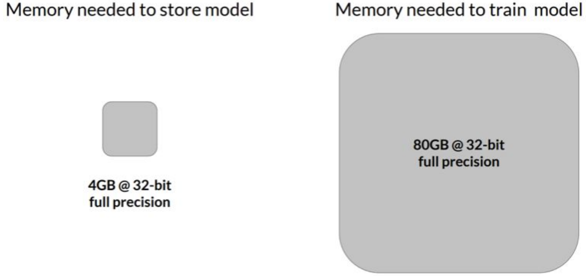
1 parameter = 4 bytes (32-bit float)
1B parameters = 4×10^9 bytes = 4GB

Additional GPU RAM needed to train 1B parameters

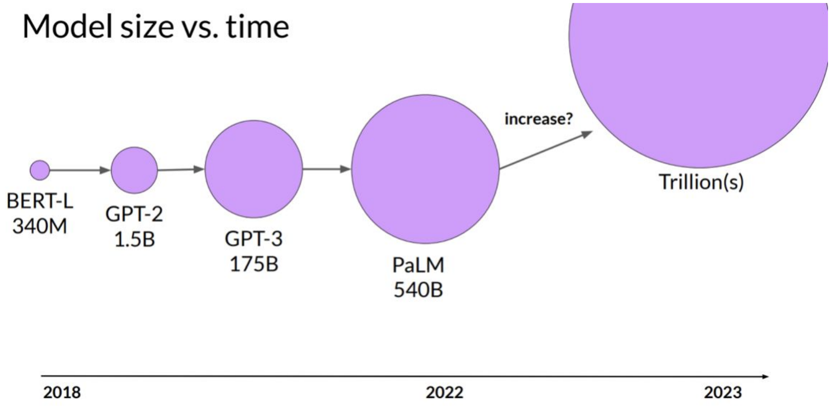
	Bytes per parameter
Model Parameters (Weights)	4 bytes per parameter

~20 extra bytes per parameter

Approximate GPU RAM needed to train 1B-params



Model size vs. time



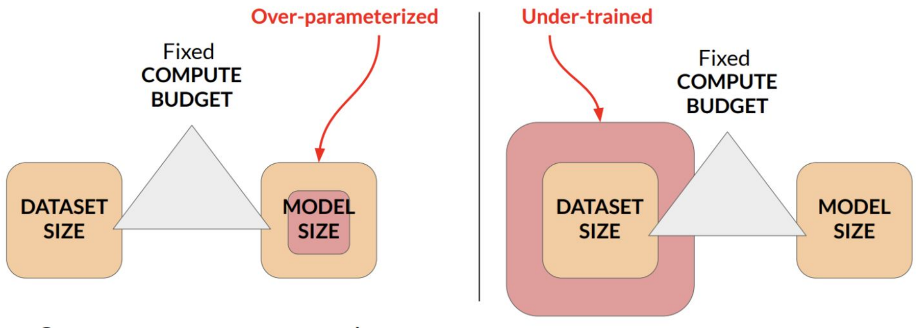
Computational challenges

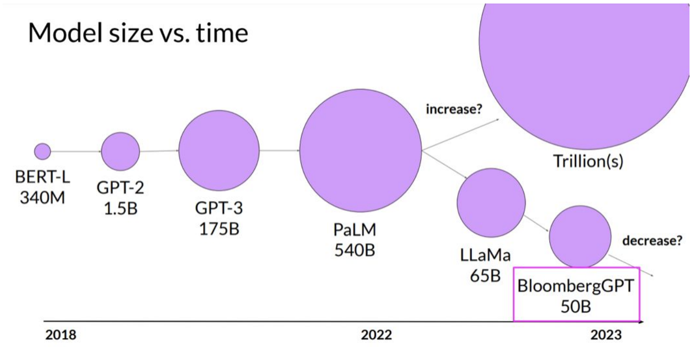
`OutOfMemoryError: CUDA out of memory.`

Motivation: Part 2

Compute resource constraint:

- Hardware
- Project timeline
- Financial Budget





Objectives

1. Understanding the multiprocessing from basics using cuda in Python
2. Picking up a simple task such as a machine translation using transformers model and perform an ablation studies with computational restraints to understand how dealing with the problem of overparameterization can lead to saving resources in multiprocessing settings

Data Distribution: CPU vs GPU

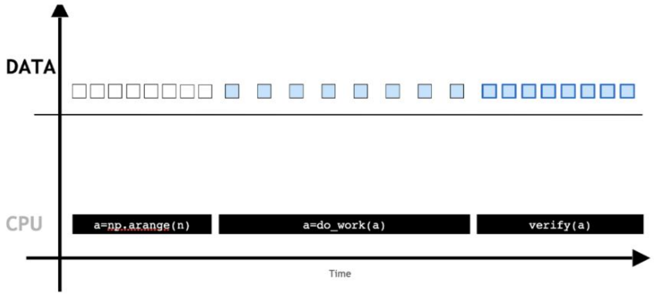


Figure 1: CPU-Accelerated: Data is allocated on the CPU and work is performed serially on the CPU

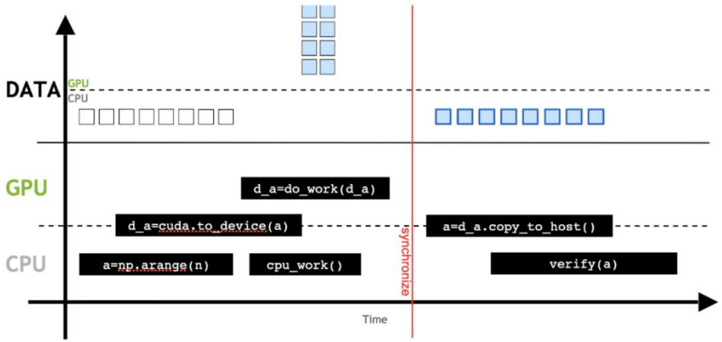


Figure 2: GPU-Accelerated: In accelerated applications both host and device memory are used, data can be initialized on the CPU and copied to the GPU device where it can be worked on in parallel. GPU work is asynchronous to the host so both CPU and GPU work can happen simultaneously. Synchronization points between CPU and GPU can be indicated using cuda.synchronize() function - data is copied back to the CPU.

Accelerated Python Experiments

Function	CPU/Python	Time Taken
Hypotenuse	Python version	693 ns ± 7.88 ns/loop
Hypotenuse	CPU	190 ns ± 0.0269 ns/loop
Hypotenuse	in-built Python	139 ns ± 0.031 ns/loop

Table 1: Hypotenuse function on CPU accelerated/Python

Function	CPU/Python	Time Taken
Scalar Addition	Host device	693 ns ± 7.88 ns/loop
Scalar Addition	CUDA functions	567 µs ± 354 ns/loop
Scalar Addition	CUDA out device	455 µs ± 623 ns/loop

Table 3: Hypotenuse function on CPU accelerated/Python

Function	Numpy/Numpy	Time Taken
Addition	Numpy on CPU	1.03 µs ± 0.24 ns/loop
Addition	Numba on GPU	688 µs ± 1.05 µs/loop

Table 2: Addition function on Numpy/Numba

Grid Stride Loop

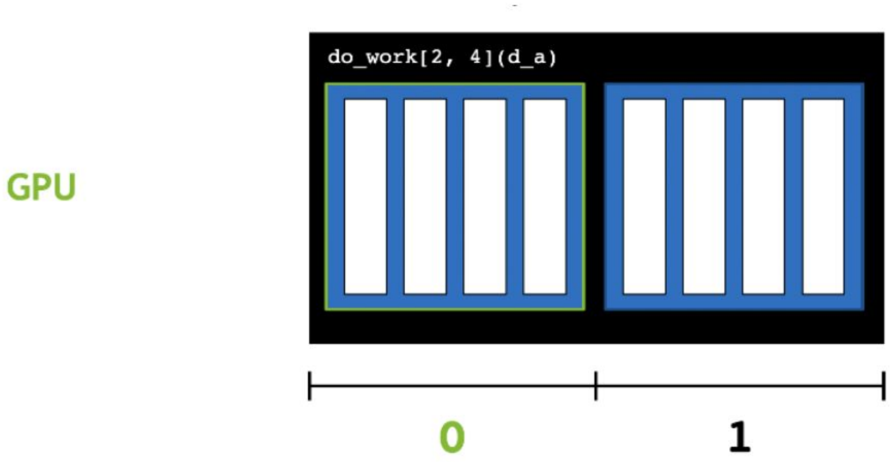


Figure 3: CUDA Thread Hierarchy Variables

Function	CPU/GPU	Time Taken
Monte Carlo Pi	CPU	108 ms ± 18 µs/loop
Monte Carlo Pi	GPU	1.05 ms ± 76.5 µs/loop

Table 4: Hypotenuse function on CPU accelerated/Python

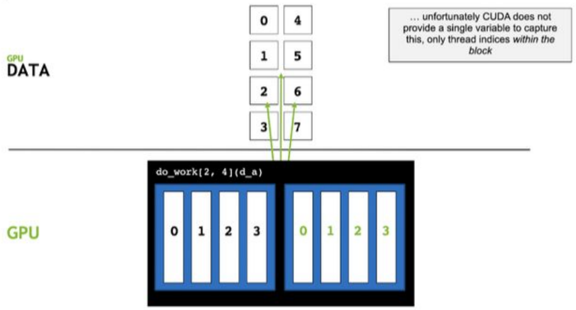


Figure 4: Coordinating Parallel Threads

Experimental Details

Task: Machine Translation from English to Spanish

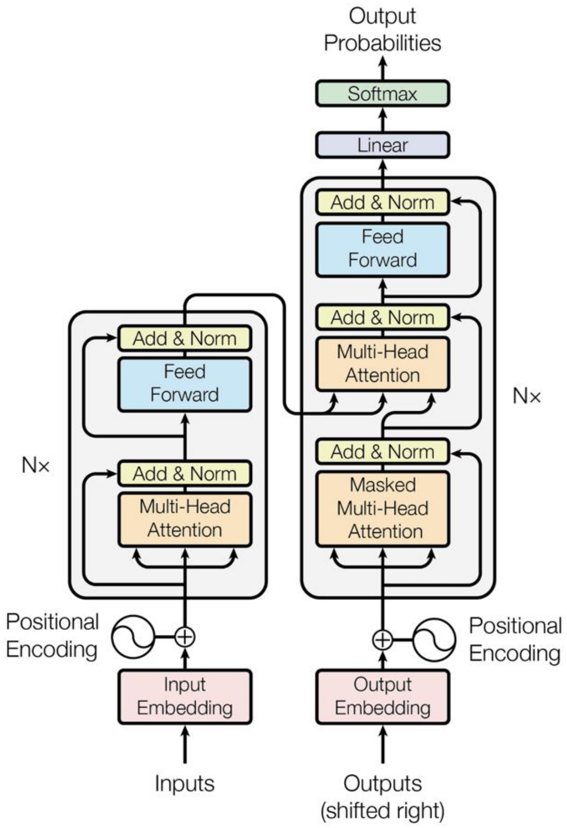
Data: spa (English to Spanish)

Train Data: 1 million translations

Training: 70 %

Validation: 30%

Architecture: Complete Encoder/Decoder Pipeline



Experimental Details

Parameters to ablate on: Embedding dimensions (referred to as model size), number of heads in one multi-head attention encoder/decoder block; number of multi-head attentions in encoder/decoder block; Dropouts; Epochs (limited to 100 for ablation). Went to 400 for checking overfitting.

Resources: 8 hours of run time, single a100 GPU with 32 cores and 64 gb ram.

Metrics: Validation Loss/ Perplexity (did not consider BLEU score) as that would have not lead to developing an understanding of what is happening per epoch.

Results with model size 16 on CPU

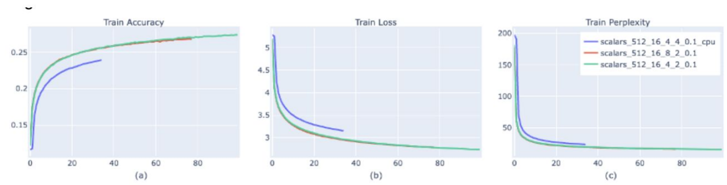


Figure 6: Experiments: Blue curve represents cpu run (34 epochs). Maximum length = 512, Model Size = 16, number of heads = 4, 8, number of layers = 2, 4 and dropout sizes = 0.1 (combinations shown in legends). Ran for 100 epochs (x-axis). **(a)** Train Accuracy. **(b)** Train Loss. **(c)** Train Perplexity.

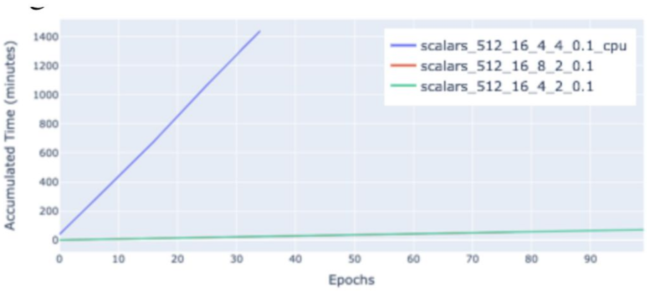


Figure 7: Experiments: Blue curve represents cpu run (34 epochs). Curves represent accumulated time in minutes vs. epochs. 34 epochs of run in CPU took 1400 minutes in comparison to only around 120 minutes for GPU runs.

Different Hyperparameters Performance

Model	Heads	Layers	Dropout	Time (min)	Train Loss	Val Loss	Val Perplexity	Parameters (million)
16	4	2	0.1	71.0	2.7389	2.3684	10.6806	3.20
16	4	4	0.1	93.0	2.9223	2.5788	13.1816	3.21
16	8	2	0.5	70.0	4.3441	4.477	87.9667	3.20
16	4	2	0.5	71.0	4.4678	6.1332	460.9081	3.20
32	4	4	0.1	98.0	1.8019	1.5429	4.6783	6.42
32	4	2	0.1	67.0	1.669	1.4727	4.361	6.36
32	4	2	0.5	67.0	3.7191	10.8591	5.2e+04	6.36
16	4	2	0.5	71.0	4.4678	6.1332	460.9081	3.20
32	8	2	0.5	68.0	3.6843	5.202	181.6349	6.36
64	4	8	0.5	244.0	5.2703	8.035	3.1e+03	13.48
64	8	8	0.5	176.0	5.0072	9.399	1.2e+04	13.48
64	16	4	0.5	98.0	4.5781	6.232	508.7538	13.01
64	4	2	0.4	71.0	2.2984	1.9782	7.23	12.78
64	4	2	0.3	71.0	1.7956	1.5268	4.6035	12.78
64	4	4	0.4	94.0	3.1409	3.3198	27.6538	13.01
64	4	4	0.3	92.0	2.0916	1.7769	5.9116	13.01
64	4	2	0.5	121.0	2.9436	7.4402	1.7e+03	12.78
128	4	2	0.1	79.0	0.5859	0.9217	2.5136	25.95
128	**4**	**4**	**0.1**	**107.0**	**0.6481**	**0.8993**	**2.4579**	**26.87**
128	8	2	0.5	79.0	2.4338	2.9636	19.3666	25.95
128	8	4	0.5	110.0	5.062	11.2901	8.0e+04	26.87
128	4	8	0.5	178.0	5.2745	11.5297	1.0e+05	28.73
128	4	4	0.5	106.0	5.2141	12.1386	1.9e+05	26.87
128	4	8	0.5	178.0	5.2745	11.5297	1.0e+05	28.73
128	4	4	0.5	106.0	5.2141	12.1386	1.9e+05	26.87
256	4	2	0.1	94.0	0.3654	1.0448	2.8427	53.67
256	4	4	0.1	120.0	1.4242	1.9795	7.2392	57.36
256	4	2	0.4	94.0	2.036	2.5577	12.906	53.67
256	4	2	0.3	93.0	1.6566	1.8869	6.599	53.67
256	4	2	0.5	94.0	2.2395	2.9339	18.8001	53.67
256	16	8	0.5	209.0	5.2948	11.2313	7.5e+04	64.73
256	4	16	0.5	314.0	5.2841	12.7906	3.6e+05	79.47
256	4	16	0.5	314.0	5.2841	12.7906	3.6e+05	79.47
512	4	4	0.5	166.0	5.3473	14.5356	2.1e+06	129.34
512	4	2	0.5	229.0	2.9624	3.4459	31.371	114.62
512	4	2	0.4	135.0	2.6893	3.4579	31.7515	114.62
512	4	2	0.3	133.0	2.5512	2.7891	16.2657	114.62

Figure 8: An analysis of the impact of different hyperparameters on machine translation model performance, along with the corresponding parameter count for each configuration (for 100 epochs)

Results with model size 512

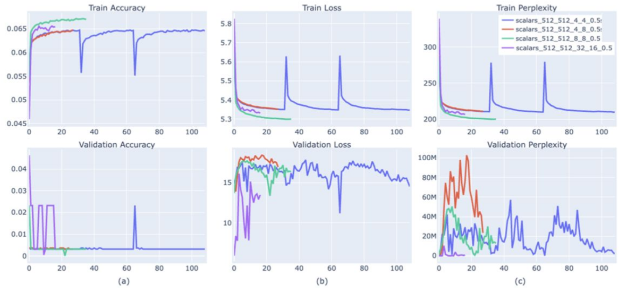


Figure 9: Experiments: Maximum length = 512, Model Size = 512, number of heads = 4, 8, 32, number of layers = 4, 8, 16, and dropout sizes = 0.5 (combinations shown in legends). (a) Train and Validation Accuracy: Comparison of training and validation accuracy across different configurations. (b) Train and Validation Loss: Plot of training and validation loss for different models. (c) Train and Validation Perplexity: Visualization of training and validation perplexity under various hyperparameter settings.

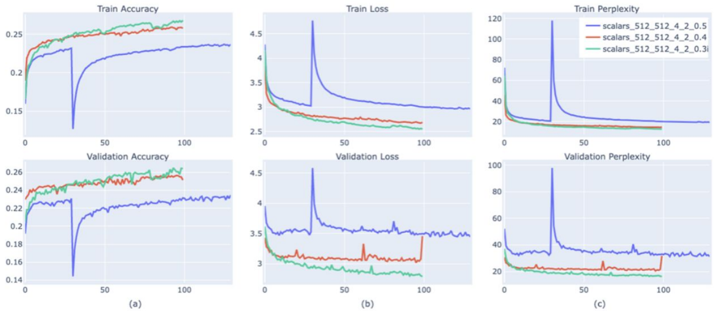


Figure 10: Experiments: Maximum length = 512, Model Size = 512, number of heads = 4, number of layers = 2, and dropout sizes = 0.3, 0.4, 0.5 (combinations shown in legends). (a) Train and Validation Accuracy. (b) Train and Validation Loss. (c) Train and Validation Perplexity.

Results with model size 256

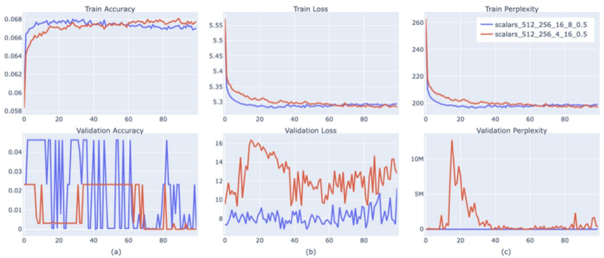


Figure 11: Experiments: Maximum length = 512, Model Size = 256, number of heads = 4, 16, number of layers = 8,16, and dropout sizes = 0.5 (combinations shown in legends). (a) Train and Validation Accuracy. (b) Train and Validation Loss. (c) Train and Validation Perplexity.

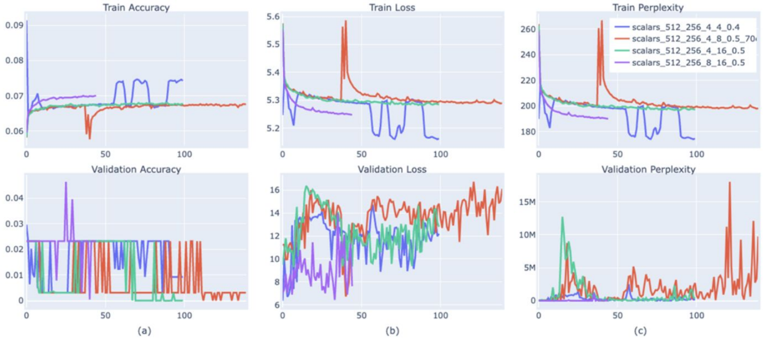


Figure 12: Experiments: Maximum length = 512, Model Size = 256, number of heads = 4, 8, number of layers = 4, 8, 16, and dropout sizes = 0.4, 0.5 (combinations shown in legends). (a) Train and Validation Accuracy. (b) Train and Validation Loss. (c) Train and Validation Perplexity.

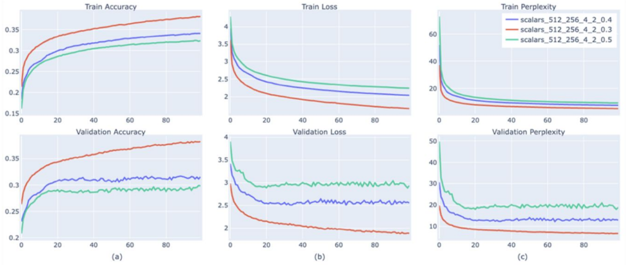


Figure 13: Experiments: Maximum length = 512, Model Size = 256, number of heads = 4, number of layers = 2, and dropout sizes = 0.3, 0.4, 0.5 (combinations shown in legends). (a) Train and Validation Accuracy. (b) Train and Validation Loss. (c) Train and Validation Perplexity.

Results with model size 128

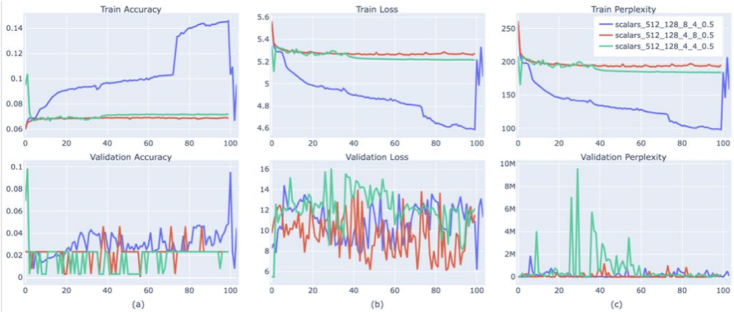


Figure 14: Experiments: Maximum length = 512, Model Size = 128, number of heads = 4, 8, number of layers = 4, 8, and dropout sizes = 0.5 (combinations shown in legends). (a) Train and Validation Accuracy. (b) Train and Validation Loss. (c) Train and Validation Perplexity.

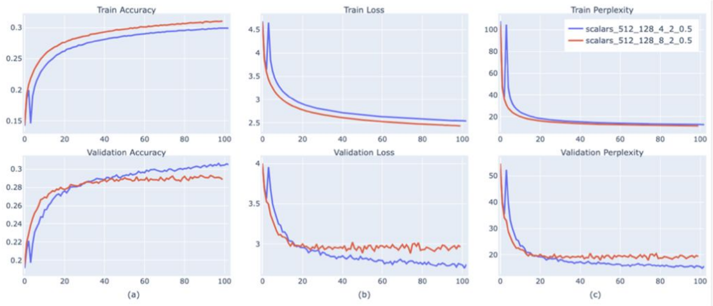


Figure 15: Experiments: Maximum length = 512, Model Size = 128, number of heads = 4, 8, number of layers = 2, and dropout sizes = 0.5 (combinations shown in legends). (a) Train and Validation Accuracy. (b) Train and Validation Loss. (c) Train and Validation Perplexity.

Results with model size 64

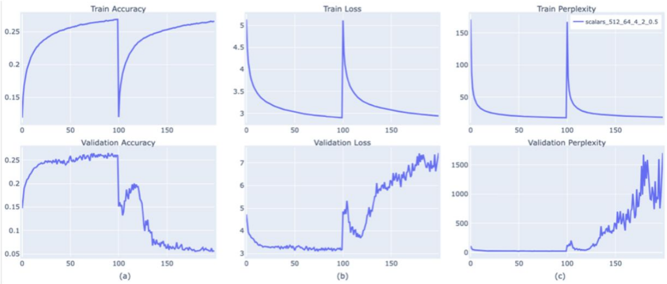


Figure 16: Experiments: Maximum length = 512, Model Size = 64, number of heads = 4, number of layers = 2, and dropout sizes = 0.5 (combinations shown in legends). (a) Train and Validation Accuracy. (b) Train and Validation Loss. (c) Train and Validation Perplexity.

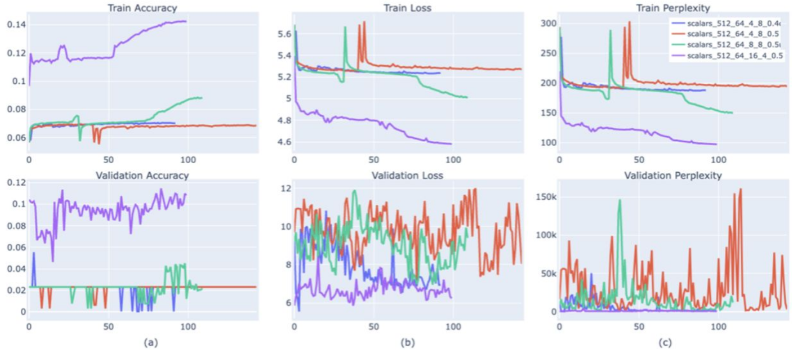


Figure 17: Experiments: Maximum length = 512, Model Size = 64, number of heads = 4, 8, 16, number of layers = 4, 8, and dropout sizes = 0.5 (combinations shown in legends). (a) Train and Validation Accuracy. (b) Train and Validation Loss. (c) Train and Validation Perplexity.

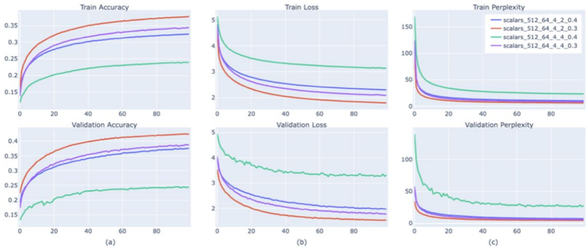


Figure 18: Experiments: Maximum length = 512, Model Size = 64, number of heads = 4, number of layers = 2, 4, and dropout sizes = 0.3, 0.4 (combinations shown in legends). (a) Train and Validation Accuracy. (b) Train and Validation Loss. (c) Train and Validation Perplexity.

Results with model size 32

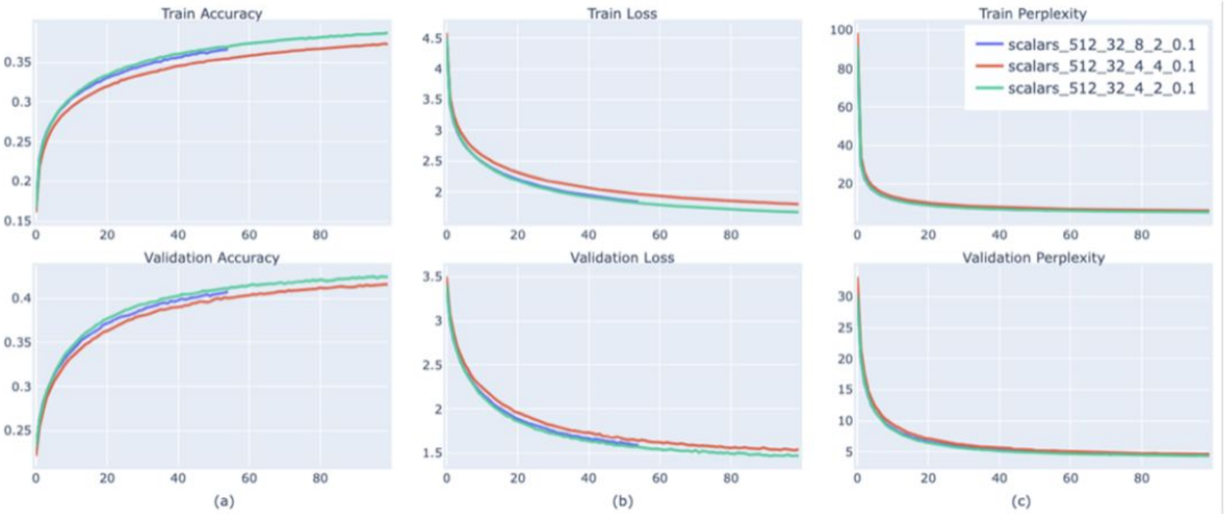


Figure 20: Experiments: Maximum length = 512, Model Size = 32, number of heads = 4, 8, number of layers = 2, 4 and dropout sizes = 0.1 (combinations shown in legends). (a) Train and Validation Accuracy. (b) Train and Validation Loss. (c) Train and Validation Perplexity.

Results with model size 16

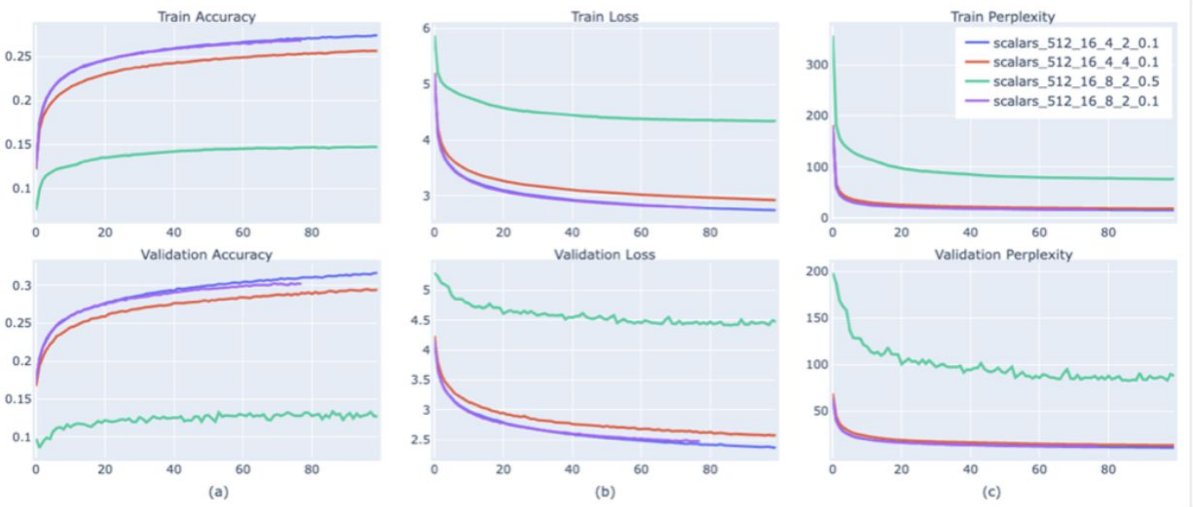


Figure 21: Experiments: Maximum length = 512, Model Size = 16, number of heads = 4, 8, number of layers = 2, 4 and dropout sizes = 0.1, 0.5 (combinations shown in legends). (a) Train and Validation Accuracy. (b) Train and Validation Loss. (c) Train and Validation Perplexity.

Best Results (After reducing dropout to 0)

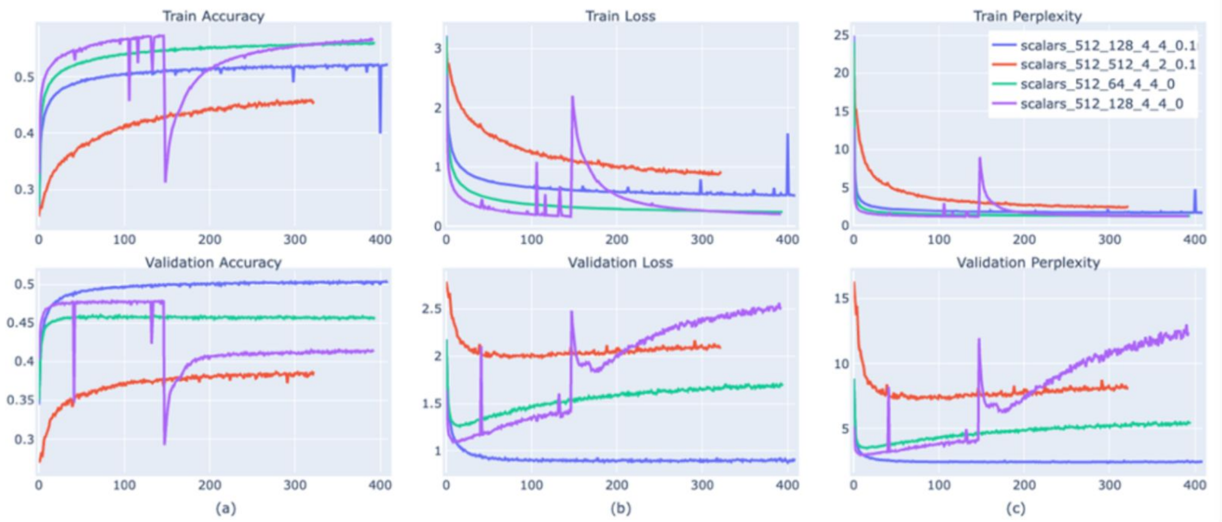


Figure 23: Experiments: Maximum length = 512, Model Size = 64,128, 512, number of heads = 4, number of layers = 2, 4 and dropout sizes = 0.1, 0 (combinations shown in legends). (a) Train and Validation Accuracy. (b) Train and Validation Loss. (c) Train and Validation Perplexity.

Conclusion

- Python Accelerated: CPU vs GPU
- Role of Dropout
- Model Complexity vs. Efficiency
- Avoidance of Excessive Compute Power
- Overfitting Trends
- Influence of Heads and Layers
- Evaluation Metrics and Perplexity
- Advocacy for Thoughtful Tuning

Thank you!