The highlighted red are documents in the zip files with the folder having the same name as the corresponding problems.

Problem1 Deliverables:

The log of these experiments, the model checkpoint and your code for training and inference. Describe modifications made to the original codebase, and any challenges faced.

Modifications made to the original code base: added the attention masking to model.py in the forward function of CausalSelfAttention

```
att = None
if pad_mask is not None:
  att = (pad_mask == 1)[:, None, None, :]  # True for pad tokens
```

One challenge we faced was to figure out the masking logic.

Problem 2.1 Deliverables:

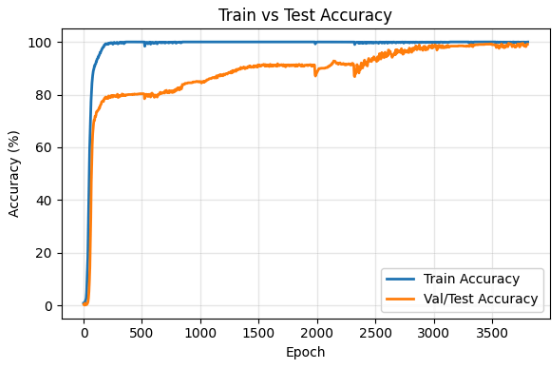
Generated train/test splits. Include description of the process and the number of datapoints used for each split.

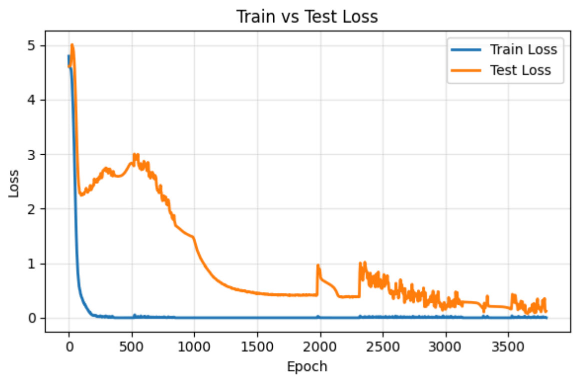
The data generation process for this project involves systematically creating all possible arithmetic equations for the operators "+", "-", and "/" over the moduli 113 and 97. For each operator and each modulus, every valid pair of operands (a, b) in the range 0 to p-1 (where p is the modulus, either 113 or 97) is considered, with division skipping cases where b is zero. For addition and subtraction, this results in 12,769 equations each for modulus 113 (113 × 113) and 9,409 equations each for modulus 97 (97 × 97), while division yields 12,656 equations for modulus 113 (113 × 112) and 9,312 equations for modulus 97 (97 × 96). The full set of equations for each operator and modulus is shuffled using a fixed random seed to ensure reproducibility. The shuffled data is then split into training (80%), validation (10%), and test (10%) sets. Specifically, for addition and subtraction, there are 10,215 training, 1,277 validation, and 1,277 test examples for modulus 113, and 7,527 training, 941 validation, and 941 test examples for modulus 97; for division, there are 10,124 training, 1,266 validation, and 1,266 test examples for modulus 113, and 7,449 training, 931 validation, and 932 test examples for modulus 97. Each split is saved as a separate text file in the data directory, named according to the operator, modulus, and split (e.g., "addition_113_train.txt", "addition_97_val.txt", "division_113_test.txt").

Problem 2.2 Deliverables:

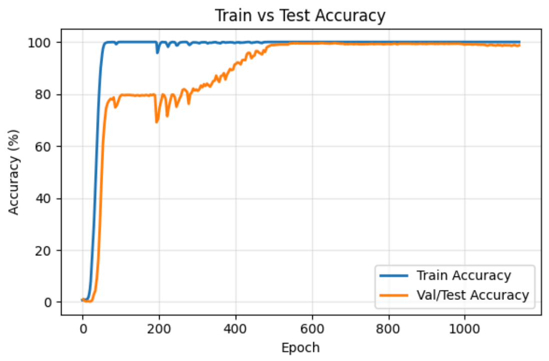
Training curves and test curves, final model checkpoint for one of the seeds. Report the final loss and train/test accuracy across 3 random seeds.

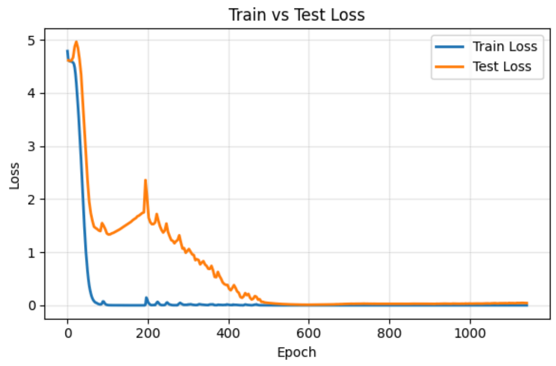
Plot for seed 0, p=97, operator + layer 1:



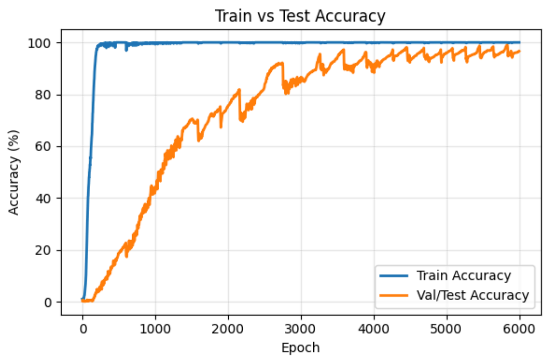


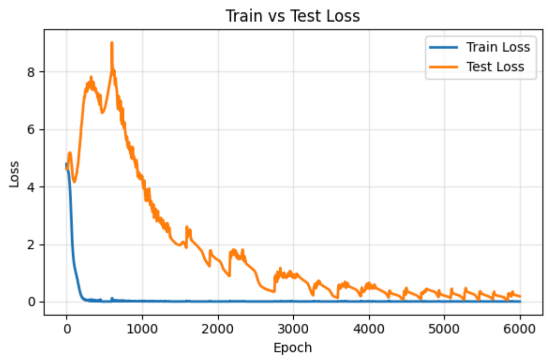
Plot for seed 0, p=97, operator + layer 2:



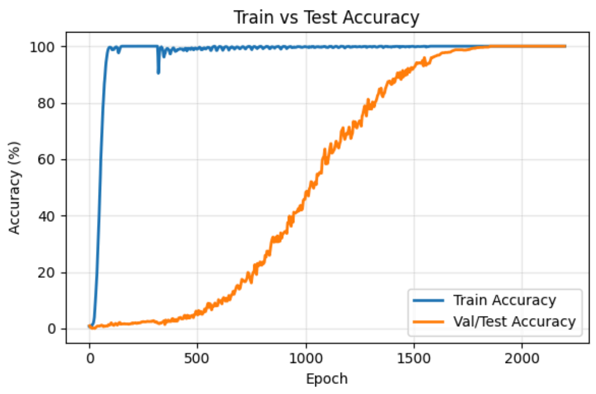


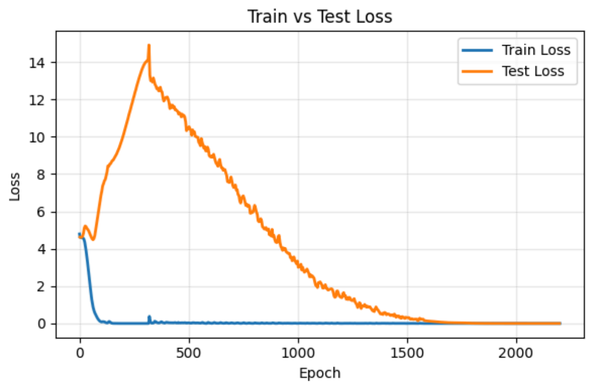
Plot for seed 0, p=97, operator - layer 1:



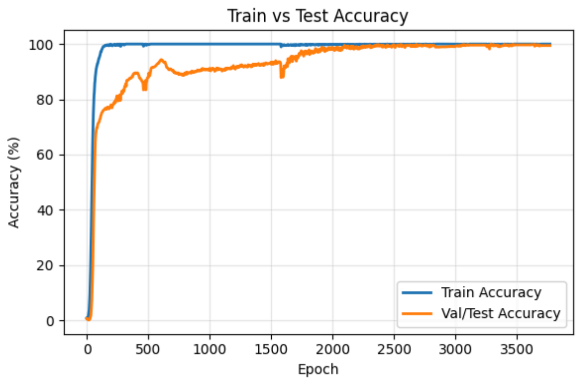


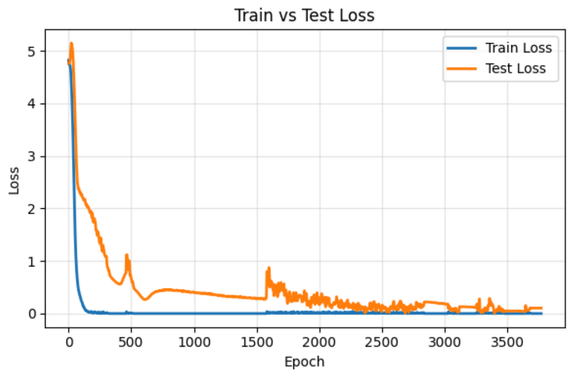
Plot for seed 0, p=97, operator - layer 2:



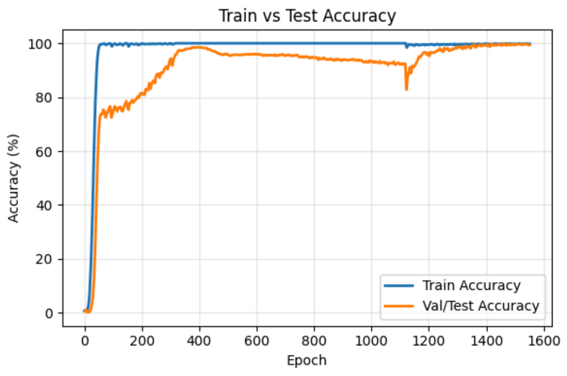


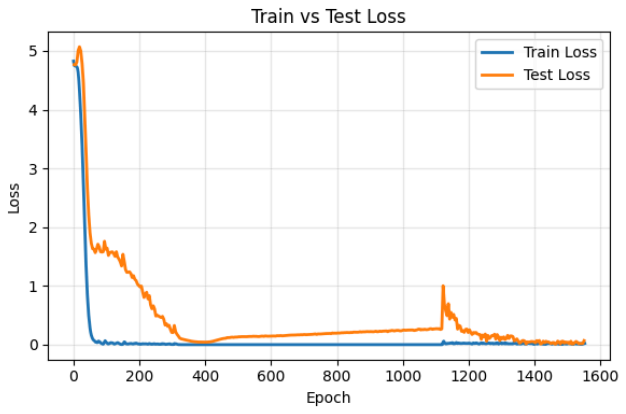
Plot for seed 0, p=113, operator + layer 1:



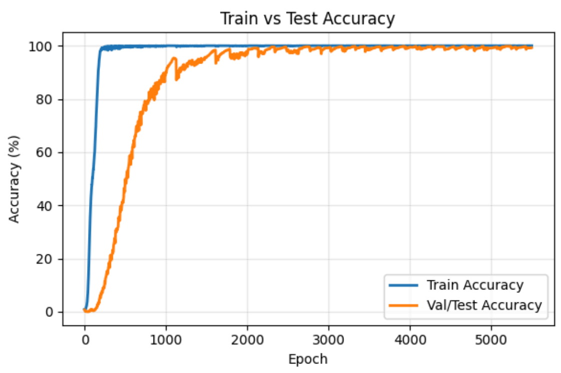


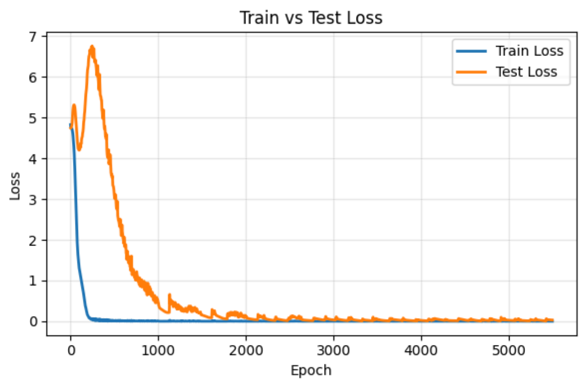
Plot for seed 0, p=113, operator + layer 2:



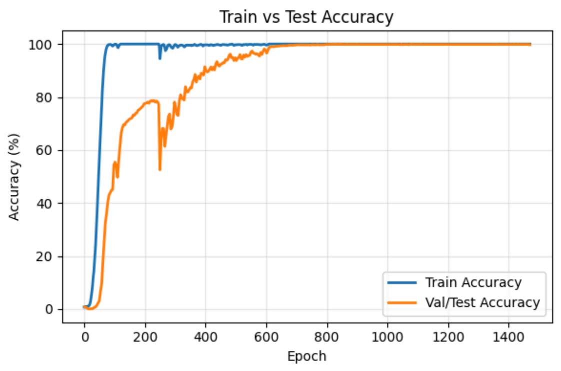


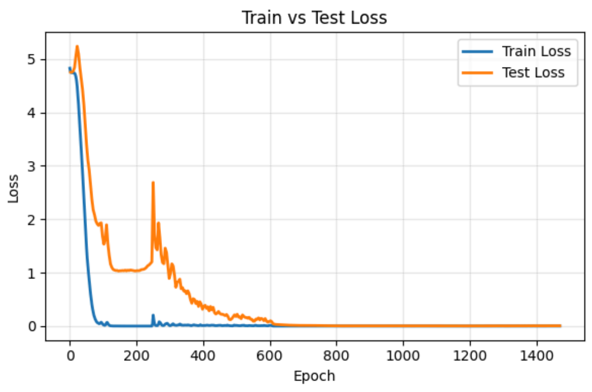
Plot for seed 0, p=113, operator - layer 1:





Plot for seed 0, p=113, operator - layer 2:





p=	Operator	Seed	Layer #	Train accuracy	Train loss	Test accuracy	Test loss
97	+	0	1	99.9867	0.0008	99.1507	0.1215
97	+	42	1	100	8.720e-08	99.2569	0.0400
97	+	599	1	99.9336	0.0045	99.6815	0.0437
97	+	0	2	100	1.330e-07	98.7261	0.0423
97	+	42	2	99.8539	0.0059	98.3015	0.1100
97	+	599	2	99.6944	0.0204	98.9384	0.0779

p=	Operator	Seed	Layer #	Train accuracy	Train loss	Test accuracy	Test loss
97	-	0	1	100	2.270e-08	96.6030	0.1782
97	-	42	1	100	8.426e-09	98.1953	0.0820

97	-	599	1	99.9601	0.0014	95.8599	0.1980
97	-	0	2	100	1.902e-07	100	0.0008
97	-	42	2	99.9336	0.0016	99.0446	0.0174
97	-	599	2	100	1.064e-07	100	0.0005

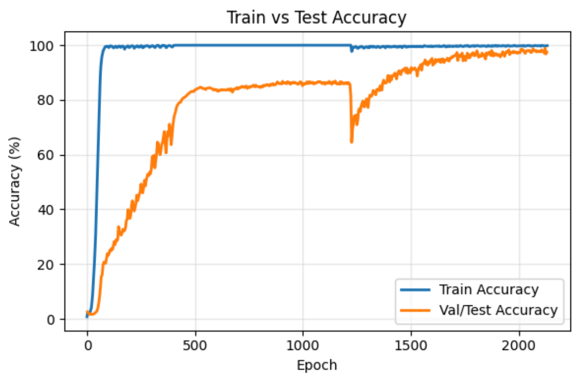
p=	Operator	Seed	Layer #	Train accuracy	Train loss	Test accuracy	Test loss
113	+	0	1	100	1.167e-11	99.4523	0.1032
113	+	42	1	100	0	99.6088	0.0624
113	+	599	1	100	9.712e-08	99.8435	0.0041
113	+	0	2	99.7259	0.0211	99.2958	0.0681
113	+	42	2	99.8532	0.0082	99.7653	0.0078
113	+	599	2	100	1.323e-07	100	8.909e-06

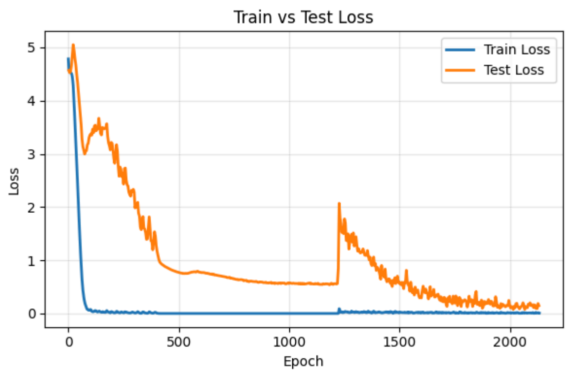
p=	Operator	Seed	Layer #	Train accuracy	Train loss	Test accuracy	Test loss
113	-	0	1	100	2.736e-08	99.2175	0.0290
113	-	42	1	100	1.392e-08	99.2958	0.0328
113	-	599	1	100	1.929e-08	98.8263	0.1475
113	-	0	2	100	1.123e-07	99.7653	0.0055
113	-	42	2	99.9315	0.0112	99.7653	0.0304
113	-	599	2	99.9021	0.0161	98.9828	0.1577

Problem 2.3 Deliverables:

Plot of training curves, final model ckpt for one seed, and instructions for inference with the model.

Plot for seed 0, p=97, operator / layer 2:





Instructions for inference with the model:

1. Load the model checkpoint

```
import torch
from grok.training import TrainableTransformer

checkpoint_path = "logs/checkpoints/last.ckpt"
model = TrainableTransformer.load_from_checkpoint(checkpoint_path)
model.eval()
```

2. Prepare the input equation. Use the same tokenizer as during training. For example, to infer 15 / 7 =:

```
input_eq = "15 / 7 ="
tokens = model.train_dataset.tokenizer.encode(input_eq)
input_tensor = torch.tensor([tokens], dtype=torch.long)
```

3. Run the model, pass the input through the model and get the output tokens:

```
with torch.no_grad():
    output, _, _ = model.transformer(input_tensor)
    predicted_tokens = output.argmax(dim=-1)
```

4. Decode the output

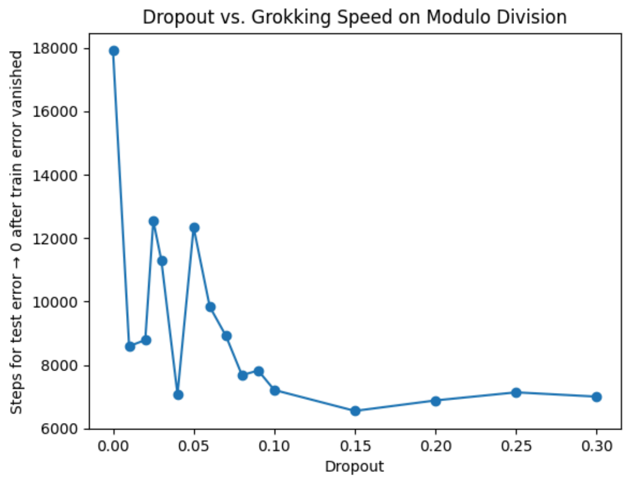
```
result = model.train_dataset.tokenizer.decode(predicted_tokens[0].tolist())
print("Predicted output:", result)
```

Problem 2.4 Deliverables:

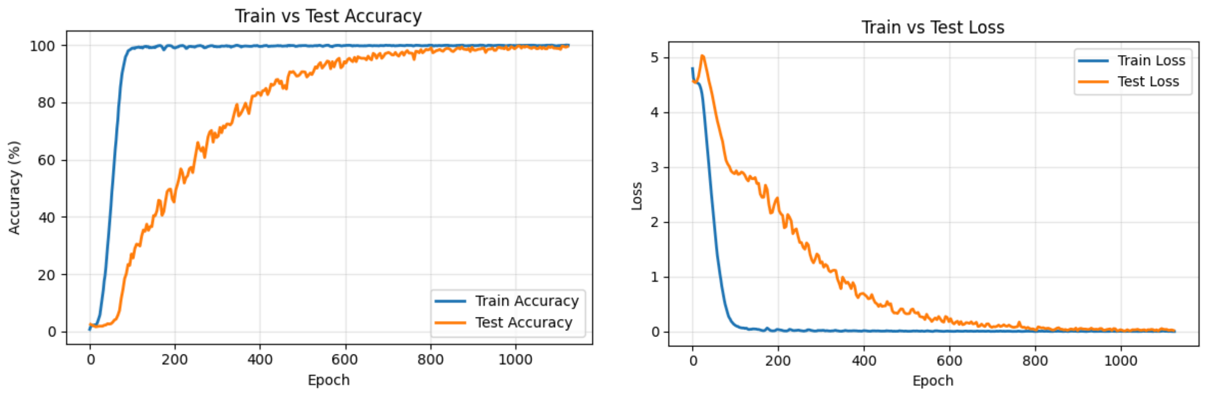
A short report of what factor you changed, and how it influenced training curves (with plots).

The factor we changed was the dropout rate. Raising the dropout rate from 0 to 0.15 means the network randomly turns off about one-fifteenth of its “neurons” on every pass. Because different parts of the model keep blanking out, it can’t rely on memorising exact training examples—the memorised link might vanish next time. Instead, it has to find a simple rule that still works even when some pieces are missing.

This pushes the model to learn the true arithmetic pattern sooner, so test accuracy starts rising much earlier and reaches the same high level with far fewer training steps.



We can see that when we increase dropout rate, the number of steps for test set to learn decreases leading to faster grokking.



The plots above are when dropout rate is 0.15, we see that at around 400 epochs, the test accuracy starts increasing compared to 500 epochs for when dropout rate is 0 (in problem 2.3).