CSE 493S/599S 2025 Spring, Homework 2

Minhak Song
Department of Mathematical Sciences, KAIST
minhak0@uw.edu

Jiwon Yun
Department of Applied Statistics, Yonsei University
jiwonyun@uw.edu

Code

Our code is available at https://github.com/supportyun/UW_CSE493_HW2.

Part 1.5 - Sanity Checks

In the `checkpoints` directory, you can find `gpt_sanity_check.pt` and `log_sanity.csv`, which contain the model checkpoint and training logs for the sanity check experiment.

Tokens were created by splitting on spaces between words. Initially, we included the sentence `"I love machine learning"` with double quotation marks, which caused an error since the tokenizer interpreted `"` as a token. It took some time to debug and fix this issue.

Aside from that, the train loss successfully converged to zero as expected, confirming the model's ability to memorize the string. To evaluate the inference performance of the model trained for the sanity check, we implemented `inference.py`. When given the input `I love machine`, the model correctly completed the sentence as `I love machine learning`.

Part 2.1 - Data Generation

The code for dataset generation is provided in `generate_data.py`. For each modulus p and operation op $\in \{+, -, /\}$, we generated all combinations of operands $a, b \in \{0, 1, \ldots, p-1\}$, excluding cases where $b = 0$ for division to avoid division by zero.

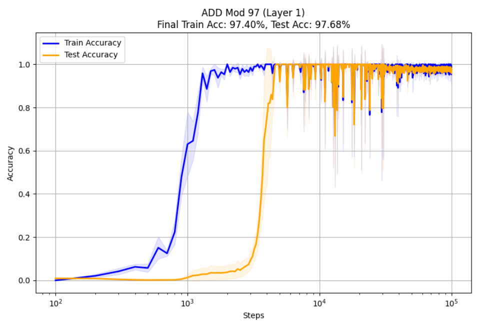
Each example follows the format `a op b = c`, where $c \equiv a \text{ op } b \pmod{p}$. The dataset is split into training and test sets using a 0.25:1 ratio.

Summary of Dataset Sizes:

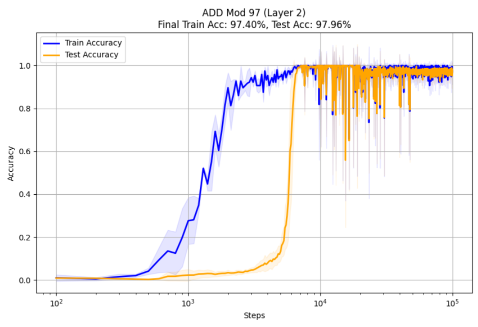
- **Addition/Subtraction mod 97:**
  - Total datapoints: 9,409
  - Train split: 2,352 examples
  - Test split: 7,057 examples
- **Division mod 97:**
  - Total datapoints: 9,312
  - Train split: 2,328 examples
  - Test split: 6,984 examples
- **Addition/Subtraction mod 113:**
  - Total datapoints: 12,656
  - Train split: 3,164 examples
  - Test split: 9,492 examples

Part 2.2 - Addition and Subtraction Experiments

We generated plots for each combination of operation (addition or subtraction) and number of transformer layers (1 or 2). Each plot includes two curves: bold lines represent the average train and test accuracy over three random seeds, and faint lines show the individual runs to visualize variance. Each model checkpoint are all stored in `checkpoints` directory.



(a) Addition Modulo 97 - layer1



(b) Addition Modulo 97 - layer2

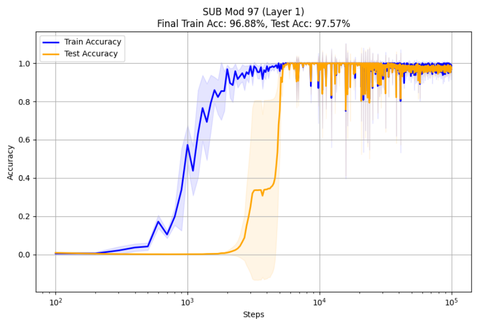
Figure 1: Train/Test Accuracy Curves for Addition (mod 97) with 1- and 2-layer GPTs

Seed	Train L	Test L	Train A	Test A
1	0.023	0.082	1.000	0.979
2	0.131	0.104	0.953	0.966
3	0.039	0.047	0.969	0.986

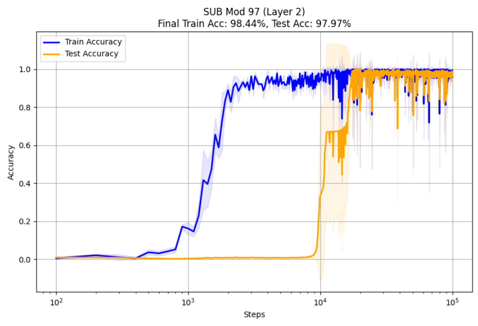
Table 1: Addition Mod 97 (Layer 1)

Seed	Train L	Test L	Train A	Test A
1	0.004	0.009	1.000	0.998
2	0.249	0.117	0.938	0.961
3	0.041	0.068	0.984	0.979

Table 2: Addition Mod 97 (Layer 2)



(a) Subtraction Modulo 97 - layer1



(b) Subtraction Modulo 97 - layer2

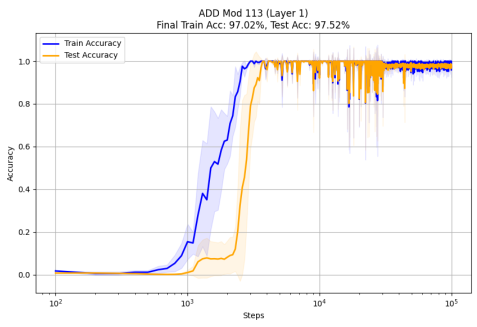
Figure 2: Train/Test Accuracy Curves for Subtraction (mod 97) with 1- and 2-layer GPTs

Seed	Train L	Test L	Train A	Test A
1	0.115	0.100	0.938	0.966
2	0.023	0.055	1.000	0.983
3	0.076	0.070	0.969	0.978

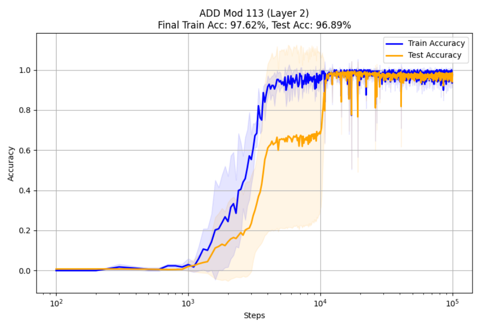
Table 3: Subtraction Mod 97 (Layer 1)

Seed	Train L	Test L	Train A	Test A
1	0.037	0.066	0.984	0.980
2	0.009	0.064	1.000	0.984
3	0.146	0.084	0.969	0.975

Table 4: Subtraction Mod 97 (Layer 2)



(a) Addition Modulo 113 - layer1



(b) Addition Modulo 113 - layer2

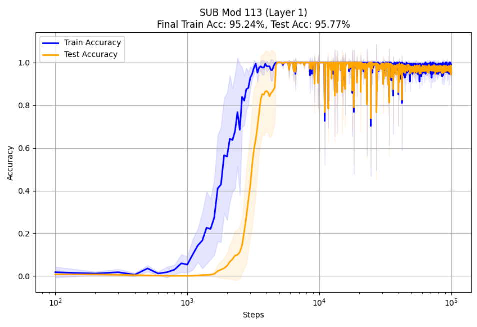
Figure 3: Train/Test Accuracy Curves for Addition (mod 113) with 1- and 2-layer GPTs

Seed	Train L	Test L	Train A	Test A
1	0.032	0.044	1.000	0.987
2	0.150	0.097	0.929	0.969
3	0.059	0.095	0.982	0.970

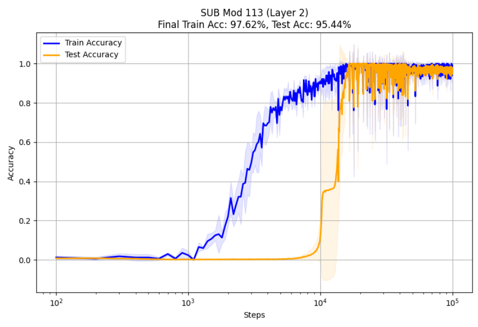
Table 5: Addition Mod 113 (Layer 1)

Seed	Train L	Test L	Train A	Test A
1	0.025	0.061	1.000	0.982
2	0.014	0.055	1.000	0.983
3	0.217	0.184	0.929	0.942

Table 6: Addition Mod 113 (Layer 2)



(a) Subtraction Modulo 113 - layer1



(b) Subtraction Modulo 113 - layer2

Figure 4: Train/Test Accuracy Curves for Subtraction (mod 113) with 1- and 2-layer GPTs

Seed	Train L	Test L	Train A	Test A
1	0.179	0.190	0.946	0.940
2	0.038	0.083	1.000	0.975
3	0.225	0.122	0.911	0.958

Table 7: Subtraction Mod 113 (Layer 1)

Seed	Train L	Test L	Train A	Test A
1	0.056	0.116	0.982	0.960
2	0.148	0.209	0.964	0.927
3	0.050	0.077	0.982	0.976

Table 8: Subtraction Mod 113 (Layer 2)

Part 2.3 Grokking

Table 9: Training Hyperparameters

Parameter	Value
BLOCK_SIZE	4
BATCH_SIZE	64
MAX_TRAIN_STEPS	10^4
NUM_HEADS	4
N_EMBD	128
FFN_DIM	512
LEARNING_RATE	1×10^{-3}
BETAS	(0.9, 0.98)
WEIGHT_DECAY	1.0
DROPOUT	0.0

Table 9 above shows the hyperparameters we used for the experiment.

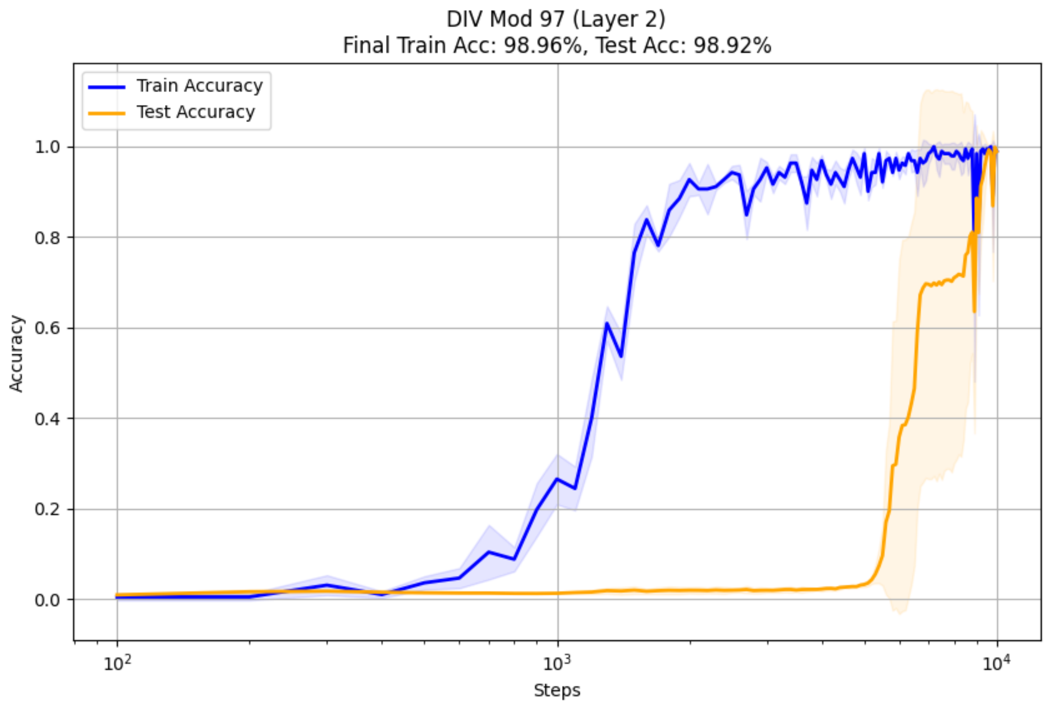


Figure 5: Division Modulo 97 - layer2

The checkpoint of this model is saved at `checkpoints/gpt_div_mod_97_seed1_layer2.pt`. We provide an inference script named `inference_problem2.py` to evaluate the trained model's performance interactively. This script loads the saved checkpoint (e.g., `checkpoints/gpt_div_mod_97_seed1_layer2.pt`), reconstructs the vocabulary from the training data, and prepares the model using the same architecture as during training.

The user is prompted to input a modular arithmetic expression such as `12/25=`. The input is tokenized, encoded using the vocabulary, and passed into the model. The model then predicts the final result token. The prediction is decoded and displayed alongside the original query (e.g., `12/25=4`).

Part 2.4 Ablation/Analysis

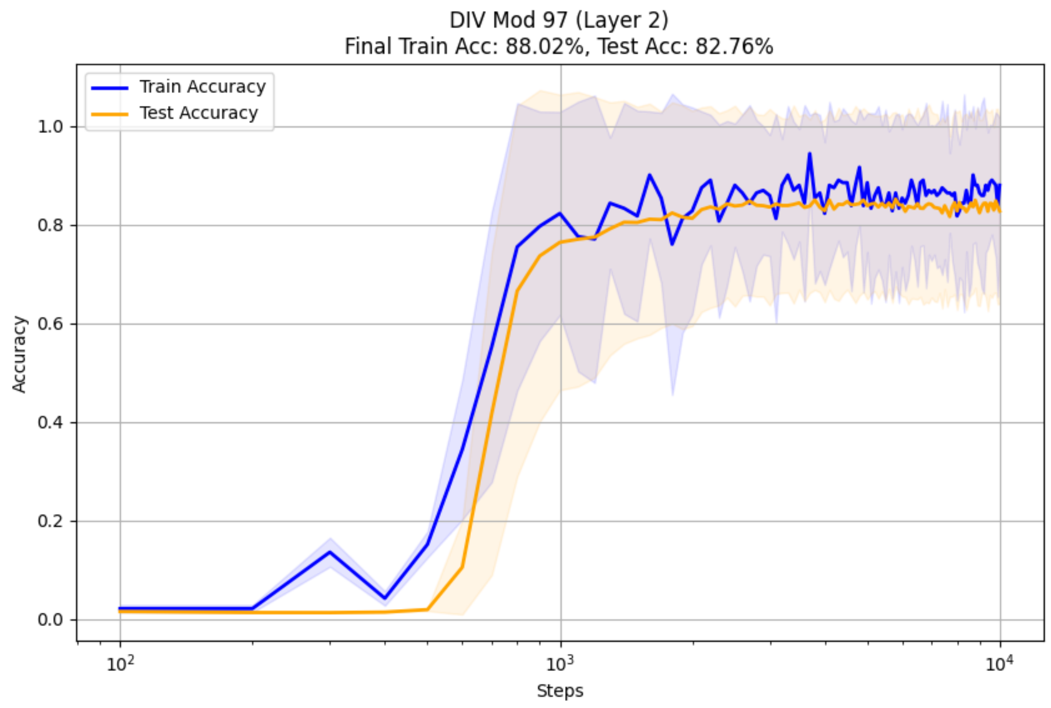


Figure 6: Division Modulo 97 - layer2 with muon optimizer(lr = 0.01)

According to the findings of Tveit et al. [2025], the Muon optimizer accelerates grokking by orthogonalizing updates to high-dimensional parameters. To evaluate its effect in our setting, we conducted an ablation study on the division modulo 97 task using a 2-layer transformer. Both experiments were trained on identical datasets for 10,000 steps, with all hyperparameters kept the same as those listed in Table 9, except for the choice of optimizer. Figure 5 uses AdamW for all parameters, whereas Figure 6 applies Muon with a learning rate of 0.01 to the hidden weights and AdamW to the remaining parameters.
Comparing the two figures, we observe a clear difference in grokking behavior. In Figure 6 shows that both train and test accuracies increase more closely together, reducing the generalization gap throughout training. This suggests that Muon accelerates generalization and narrows the memorization-generalization lag that is typical in grokking as described by Power et al. [2022].

References

Alethea Power, Yuri Burda, Harri Edwards, Igor Babuschkin, and Vedant Misra. Grokking: Generalization beyond overfitting on small algorithmic datasets. *arXiv preprint arXiv:2201.02177*, 2022.

Amund Tveit, Bjørn Remseth, and Arve Skogvold. Muon optimizer accelerates grokking. *arXiv preprint arXiv:2504.16041*, 2025.