Project 4:
Reproducibility in Data Science
A Data Ethics Project

Group 7

Richard Hoehn (**richardhoehn**)

Hector Rogel (**rogel9007)**

Isaiah Osborne(**IMOsbo**)

Karson Woods (**Kwoods132**)

Introduction

The paper we attempted to reproduce was “GROKKING: GENERALIZATION BEYOND OVERFITTING ON SMALL ALGORITHMIC DATASETS” [1]. This paper was published by OpenAI in 2022 and shows that, for at least a certain subset of problems, continuing to overfit a neural network on its training set eventually dramatically increases performance on a held-out validation set, a process the paper called “grokking” [1].

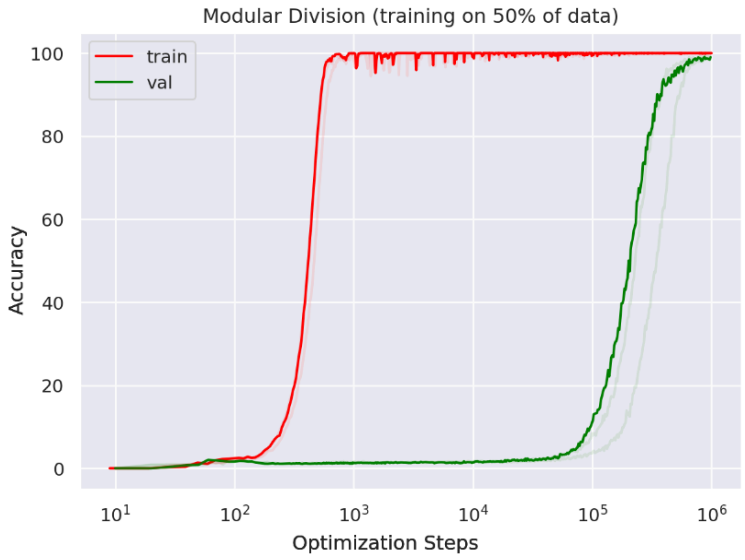


Figure 1: An example graph from the paper showing the “grokking” process

Methodology

Reproduction Approach

The following are approaches to building the code and testing the paper's claims ourselves - in essence reproducing the results of the paper's claims.

We followed the three main steps to reproducing the Grokking paper's results by:

1. Cloning Original Code
2. Setting up a dedicated environment
3. Processing / Training the Model

Cloning Original Code

The first steps to testing is making sure we do a correct job of cloning the original code from the original authors github repository. Using the methodology to add a "sub-module" in git is probably the best way to ensure that our own repository is available for future review. With this being said we used `"git submodule add https://github.com/openai/grok code/grok"`, which allowed for a clean way to keep references to the original code's repository. More details can be found from the ChatGPT thread here [2]. This creates a link to the external repository at a specific commit, allowing me to track and control updates independently of my main project. The submodule is stored in the `".gitmodules"` file, enabling reproducibility and modular organization of external code.

Environment Setup

There were some tricks that we needed to work through on setting up the environment for the grok testing. The most glaring issue the fact that OpenAI had not fixed their dependency versions in their setup code. After three years of updates, several of the dependencies involved went through breaking changes, breaking compatibility with some of PyTorch APIs used in the original Grok project. After trying unsuccessfully to determine the latest package versions for each of the dependencies released before the paper was released, we discovered that a pull request with updated package versions and API tweaks, which we were able to use to complete the environment setup. To help streamline the installation process, we created a shell script (`bootstrap.sh`), which automatically creates a Python virtual environment, clones the repo, applies the code changes from the pull request, and then installs the Grok package.

Data Processing Steps

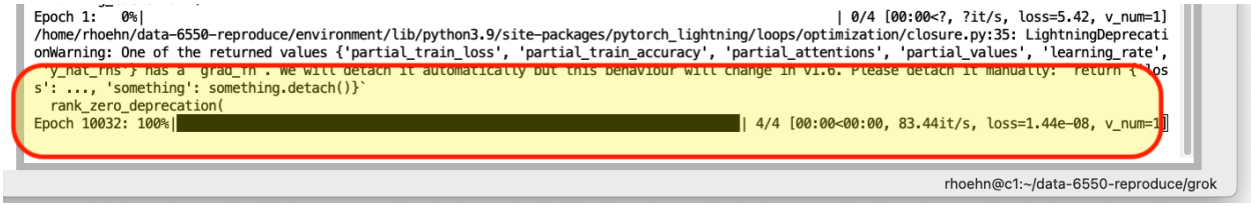
Once the environment was successfully set up, we resolved all package and configuration issues and began training our Grokking model. Based on the paper Grokking: Generalization Beyond Overfitting on Small Algorithmic Datasets by Power et al., the authors suggest training for up to

1,000,000 optimization steps (or epochs, depending on batch size and update schedule) to observe the delayed generalization effect they call grokking—where a model, after severely overfitting, suddenly begins to generalize.

They noted that while training accuracy can become nearly perfect after just ~1,000 updates, it may take up to 1,000,000 steps before validation accuracy improves—indicating generalization.

Following this guidance, we used a basic GPU (as referenced in the paper on Hamilton) and began training on MTSU's high-performance computing cluster. Our preliminary run reached 10,000 epochs in approximately 5–6 minutes. Extrapolating this, completing 1,000,000 epochs would take around 550 minutes, or approximately 9.1 days of continuous computation. Unfortunately, the HPC environment limits job durations to 48 hours, so we implemented a recurring 48-hour job cycle.

Using this approach, we were able to surpass 50,000 epochs and collect corresponding loss values. The loss values were stored during the training process in a metrics.csv file that was used to plot our train and validation losses and compare to the authors results.



Results

In this section we review our “reproducibility results” and how those affected our analysis.

Comparison with Original

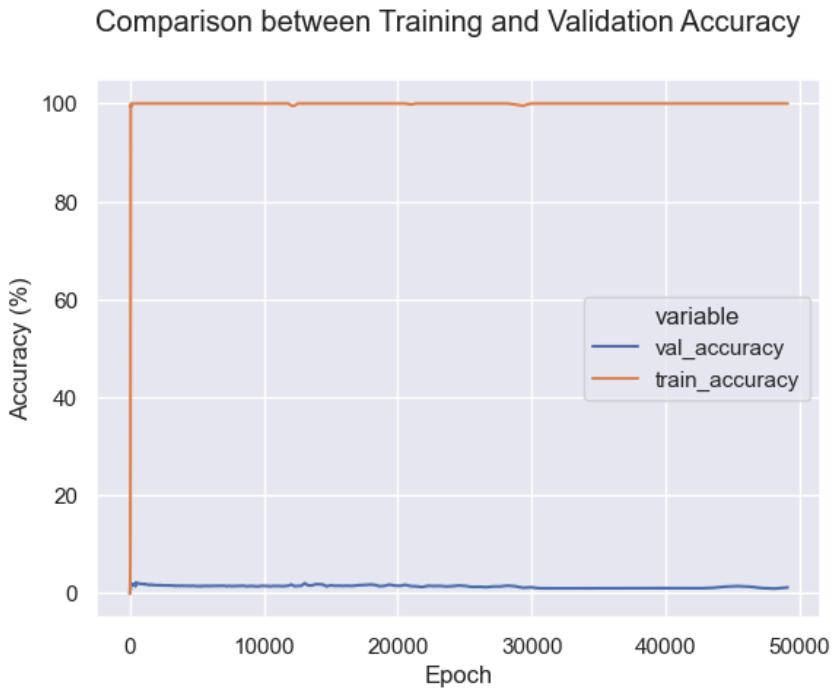


Figure 3: Trends in our training and validation accuracy

Unfortunately, our model did not perform nearly as well as expected by the Grokking paper. Despite training our model overnight for over 50000 iterations, we were not able to show any meaningful increase in our validation accuracy, instead showing a much more typical flat validation accuracy trend. While this does not necessarily disprove the paper, it is still concerning that we were not able to see the hoped-for effects.

Discrepancy Analysis

The major discrepancy is the fact we cannot train our model for as long as the paper suggests to be able to see any differences. This seems like an oversight from the paper perspective to not list that training must have taken an enormous amount of time in the paper’s year of writing in 2019! This being said, we are confident on the initial metric results we got that are related closely to those of the paper, however we were not able to see that validation generalization started to improve around 1,000,000 epochs.

Ethics Discussion

Reproducibility Challenges

As mentioned in the prior results section, the time it would take us to reproduce the training of the Grokking paper made it difficult to reproduce the results. Overall the paper states that it can be reproduced by a single GPU, which is correct, however at the cost of running for multiple days.

Obviously this poses a real challenge to any reproduction since the barrier to even get to just the data validation part takes over 9 days of processing, which in many circumstances is prohibitive for the general scientific public.

Resource Considerations

Per the paper, the resources needed to conduct this experiment are minimal. They claim that this experiment can be run on a single GPU; as evidenced below, we could run it on a single GPU. Expanding upon this further, only 16% of the graphics card power was being used in this particular instance, reinforcing the authors' claim that the resources needed for this experiment are minimal. The experiment also utilizes small algorithmically generated data sets, which contributes to the low resource requirement.

```
[rhoehn@c2 version_1]$ nvidia-smi
Tue Apr  1 17:19:44 2025
+-----------------------------------------------------------------------------------------+
| NVIDIA-SMI 550.54.15              Driver Version: 550.54.15      CUDA Version: 12.4     |
|-----------------------------------------+------------------------+----------------------+
| GPU  Name                 Persistence-M | Bus-Id          Disp.A | Volatile Uncorr. ECC |
| Fan  Temp   Perf          Pwr:Usage/Cap |           Memory-Usage | GPU-Util  Compute M. |
|                                         |                        |               MIG M. |
|=========================================+========================+======================|
|   0  NVIDIA GeForce RTX 2080 Ti     On  |   00000000:68:00.0 Off |                  N/A |
|  0%   46C    P2             59W /  250W |     643MiB /  11264MiB |     16%      Default |
|                                         |                        |                  N/A |
+-----------------------------------------+------------------------+----------------------+
```

Figure 4: Resource Usage

Scientific Integrity Implications

Considering we were able to reproduce the initial metrics but not the validation generalization the original paper achieved/presented, there are some questions about the scientific integrity of this paper. Some things to consider include the validity of these results and potential issues with transparency and documentation.

Regarding the validity of these results, while we were able to reproduce the initial metrics and are reasonably confident in them, we likely couldn't reproduce the validation generalization due to not being able to run the same number of epochs. Not being able to reproduce these results

brings into question the validity of the paper's results. It's possible that even if we ran the same number of epochs as the original paper, we still would not be able to reproduce their results.

Looking at potential issues with transparency and documentation, as mentioned previously, there is no mention of how long the original authors' training of their model took. It would take us multiple days to run the same number of epochs, which is prohibitive in replicating their results. This ties into the original authors' mention of being able to run this on a single GPU, which is true but is misleading considering the aforementioned time needed to run the same number of epochs. It also leads to the question of what else they might not have documented that could have a large impact on the ability to reproduce their results, if they're even replicable at all.

Recommendations

Based on our hands-on experience trying to reproduce grokking, one thing is clear. Training large models with hundreds of thousands of steps on a single GPU is not feasible under tight timelines. Reproducing grokking from the original paper turned out to be more challenging than expected. We found that even scaling up batch size and optimizing GPU usage wasn't enough to offset the sheer training duration required. Instead of trying to push the process as much as possible, we recommend designing experiments around hardware constraints from the beginning. Use adaptive checkpoints to evaluate progress and pivot strategies as needed. Finally, set realistic goals. Aim for patterns or trends that point toward grokking-like behavior rather than full replication.

Best Practices

Start small and validate often. Before scaling up, make sure your code runs smoothly on a smaller model and dataset. Use clear checkpoints and logging to track your progress and catch problems early. When changing hyperparameters like batch size or learning rate, keep the rest of your pipeline stable so you can isolate effects. Monitor GPU utilization closely, and do not hesitate to iterate if performance isn't improving. Organize your logs and outputs from the beginning so it's easier to compare results across experiments.

Process Improvements

If we were starting over, we would plan our training pipeline around hardware limitations first. That means figuring out optimal batch sizes and training steps for our available vram before diving in. We would begin with a simple setup to make sure everything runs cleanly and then scale up gradually, checking performance at each stage. We would also save more frequent checkpoints to avoid losing time from unexpected interruptions. Having a better system in place for restarting and resume training would have saved us from duplicate effort. Most importantly, we would focus on building flexibility into our process, so we could pivot without starting from scratch.

Work Cited

[1] A. Power, Y. Burda, H. Edwards, I. Babuschkin, and V. Misra, “Grokking: generalization beyond overfitting on small algorithmic datasets.” arXiv, Jan. 06, 2022. doi: 10.48550/arXiv.2201.02177. Available: http://arxiv.org/abs/2201.02177. [Accessed: Apr. 02, 2025]

[2] “ChatGPT - Clone repo into subfolder,” ChatGPT. Available: https://chatgpt.com/share/67e98a51-7af0-800e-bc7f-d208caefd7bf. [Accessed: Apr. 02, 2025]