Homework 5

TAs: Baihong, Chris, Gunjan, Jacob

10-605/805: Machine Learning with Large Datasets
Due Monday, November 24th at 11:59 PM Eastern Time

IMPORTANT: Be sure to highlight where your solutions are for each question when submitting to Gradescope otherwise you will be marked 0 and will need to submit regrade request for every solution un-highlighted in order for fix it!

Note that Homework 5 consists of three programming parts. Remember to fill out the collaboration section found at the end of this homework as per the course policy.

All students are required to complete the following:

1. Programming Section 1 *[50 points]*

2. Programming Section 2 *[20 points]*

3. Programming Section 3 *[10 points]*

4. Programming Section 4 (optional) *[0 points]*

All students are required to complete sections 1 through 3 of the homework. Your homework score will be calculated as a percentage over the maximum points you can earn, which is 90.

1 Programming Part 1 [50 Points]

1.1 Introduction - Zero Shot Hyperparameter Tuning

Llama2 70B cost an estimated $3.9 million to train, GPT-4 an estimated $78 million to train, and Gemeni Ultra an estimated $191 million. (Source: https://hai.stanford.edu/ai-index/2024-ai-index-report/research-and-development, Figure 3). If training a machine learning model from scratch just once costs this much, how can we perform hyperparameter tuning?

In this homework we will consider the problem of "zero-shot" hyperparameter tuning – selecting hyperparemeters for a model which we only have sufficient budget to train to completion once. Traditional hyperparemeter tuning methods such as grid search or random search rely on being able to train several models to convergence and selecting the model with the best validation performance. This paradigm fails in the context of training massive foundation models on very large datasets, as these models can only feasibly be trained from scratch a very limited number of times.

1.2 Background

1.2.1 GPT-2 Class Models

In 2019, OpenAI released Generative Pre-Trained Transformer 2 (GPT 2), trained on a dataset containing 8 million web-pages with the objective of predicting the next word of a given sentence. The GPT-2 class of models are:

- GPT-2 Small (124 million parameters)

- GPT-2 Medium (350 million parameters)

- GPT-2 Large (774 million parameters)

- GPT-2 Extra-Large (1.5 billion parameters)

The sizes of these models scale the relative performance and computation costs. The GPT-2 class models are commonly used for tasks such as text-generation and question-answering, which you will be putting to the test in this assignment while you balance between model performance and computational cost.

1.2.2 HellaSwag Evaluation

HellaSwag is a popular LLM benchmark used to determine if LLM's have reached commonsense levels of inference. The questions in this benchmark are trivial for humans (> 95% accuracy), but LLM's have a much harder time of conducting inference that seem like common sense to humans. At the time, OpenAI's GPT-2 model struggled with solving such questions, reaching an accuracy of roughly 30%, thus one of the goals of this homework is to increase the HellaSwag accuracy of your models through the strategies that you will be devising.

1.3 Instructions

1.3.1 Overview

Your goal in this homework is to find the "max learning rate" hyperparameter which achieves minimal validation loss and maximal HellaSwag accuracy for a fixed training objective. The training objective is to pretrain from scratch a GPT-2 class model (GPT-2 Small, depth=12, channels=768, heads=12, 124M parameters) on 2.5 billion tokens of training data from the FineWeb dataset.

Unfortunately, you will be constrained to only have $\sim$ \$40 $^{??}$: sufficient budget to complete an entire pretraining run approximately twice. A naive strategy might be to simply train the model twice with two

different max learning rates and choose the candidate max learning rate with better validation performance. It is your job to instead design a methodology to use this limited budget to evaluate a much larger set of candidate max learning rates ($\gg 2$).

The approach we recommend in this homework is the following: first, scale down some aspect of the training process. Then, perform hyperparameter tuning in this scaled-down setting. Finally, in the "transfer" stage, use this information to create an estimation of the optimal hyperparameters in the full-scale model. The following are all properties of the training process that might be scaled down:

1. The number of training iterations

2. The training dataset size

3. The model size (total number of parameters)

    (a) The model depth (number of layers)
    (b) The number of attention heads
    (c) The embedding dimension
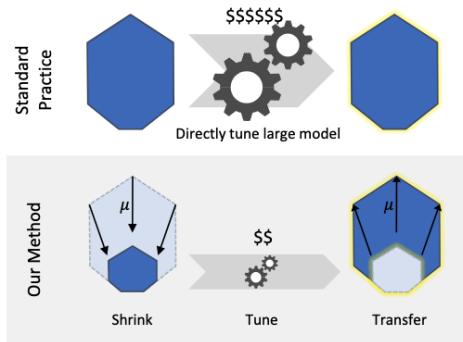

Figure 1: A visualization of using scaling to perform single-shot hyperparameter tuning. Source: Yang et al.

In addition to selecting the aspects of model training to scale down, a challenge you will face with this approach is the "transfer" stage. Namely, how do we form a prediction about the relationship between the optimal hyperparameters in a scaled-down setting vs. the full-scale setting? For example, suppose that you are investigating the scaling behavior of some parameter of the training process, Y. (as discussed, this might be the number of training iterations, the model size, or the training dataset size, etc.) For the "transfer" stage, you will want to form a hypothesis about the relationship between Y and a hyperparameter of choice α.

The following might be hypotheses that you explore:

1. No scaling. Assume that hyperparameters transfer between training processes of different scales. In other words, assume that the same hyperparameter values that are optimal in a small-scale training setting will be optimal for a large-scale training run.

2. Functional scaling. Assume that $\alpha = f(Y)$ for some function f. You might explore one of the following functional families:

    (a) Sublinear scaling
    (b) Linear scaling

3

(c) Polynomial scaling

    (d) Exponential scaling

    (e) etc.

1.3.2 Grading Criteria

You will submit your final learning rate and total hyperparameter tuning expenditure as recorded in this handout. Your final model weights will be evaluated on HellaSwag accuracy and validation loss using a held-out dataset. A small portion of your grade will depend on how close your selected learning rate is to the global optimum, with full credit given for values within a "reasonable" range, determined by past and current student performance.

1.3.3 Total Budget

We ask you not to use more than the following budget in performing your hyperparameter tuning (Part 1). **Parts 2 and 3 will require a fixed budget of \$30-\$45. Please leave at least \$40 in your account or you will be unable to complete this homework.**

1. Part 1 (hyperparameter tuning): \$20 [1]

2. Part 2 (data preprocessing): \$20-\$25

3. Part 3 (pretraining): \$10-\$20

In order to avoid accidentally exhausting your entire budget with a single training run, it is **mandatory** to perform a cost analysis of each experiment in advance. This entails doing the following:

1. Run your experiment for just 20 steps. The first (0th) step is an anomaly and is much slower due to memory loading. Get a worst-case estimate from steps 10-19, just to be safe. Record which step in the range step 10 to step 19 took the longest, and how long it took (in milliseconds).

2. Use the formula below to estimate the runtime of the full experiment in hours:

$$\texttt{Runtime (hours)} = \frac{\texttt{time per step (ms)} * \texttt{number of steps}}{3.6 \times 10^6 \texttt{ ms per hour}}$$

3. Multiply this value by the cost of running an AWS g5.xlarge per hour (\$1.063/hour at time of writing, but please round to \$1.50/hour to be safe).

1.4 Getting Started

1.4.1 Requesting GPU Access from AWS

Please check your AWS vCPU Quota for "All G and VT Instances". This should say at least "8." If you have not yet completed your AWS Quota request, please see Piazza for details on how to do so.

---

[1]If you have additional AWS credits, you may use all but the last \$40 in Programming Part 1 (as mentioned in recitation). However, please leave enough to complete Programming Part 2 and Part 3 (\$40)

1.4.2   Launching your AWS Resources

The following is a high level overview of what you will need to do to setup your AWS instance and begin running experiments. For a much more detailed guide, see the recitation materials.

- Create a S3 bucket. This will be used to store your training logs.

- Create an IAM role and user (save your access key and secret key somewhere safe!). This is required to save your training logs to an S3 bucket.

- Create a g5.xlarge instance using 'ami-08144b201b85c4460' and providing the IAM role.

1.4.3   Setting up your AWS credentials

- Type `aws configure` into the command line, and pass in your access and secret key when prompted. Make sure the region matches that of your EC2 instance (likely `us-east-1`).

1.4.4   Getting the training code

- Clone the github repo: `git clone https://github.com/10605/llm.c`

- Compile the model: `make`

1.4.5   Getting the FineWeb10B dataset

We have preloaded a tokenized subsection of the FineWeb10B dataset onto the Part 1 AMI for you to begin working with.

1.4.6   Running an Experiment

Now you are ready to run your own experiment! You can do so with the following command:

```
./train_gpt2cu \
    -o my_first_experiment \
    -b 32 \
    -x 20
```

1.5   Hyperparameters and Flags

Description	Parameter	Default Value
learning rate	—l	6e-4
output log dir	—o	`NULL, no logging`
max_steps of optimization to run	—x	-1, 1 epoch steps (default: 4725)
starting step, if training without loading optimizer	—ss	0

Table 1: Flags which you will absolutely need to use in this homework

Description	Parameter	Default Value
model depth	—ld	12
model channels	—lc	768
model number of heads	—lh	12
early stopping steps	—es	-1 (don't)
write optimization checkpoints every how many steps?	—n	0 (don't)
resume optimization found inside output log dir? (0=restart/overwrite, 1=resume/append)	—y	0, restart
input .bin filename used to continue training a model	—e	

Table 2: Flags which might be very useful, depending upon your strategy

Description	Parameter	Default Value
train data filename pattern	—id	`data/fineweb_train_*.bin`
val data filename pattern	—iv	`data/fineweb_val_*.bin`
tokenizer file path	—it	`data/gpt2_tokenizer.bin`
hellaswag file path	—ih	`data/hellaswag_val.bin`
(per-GPU, micro) batch size B	—b	64

Table 3: Flags which may be useful for solving errors

Description	Parameter	Default Value
learning rate scheduler ("cosine", "linear", "constant", or "wsd")	—k	"cosine"
learning rate warmup iterations	—u	470
learning rate decay: final fraction, at end of training	—q	0.0
weight decay	—c	0.1
total desired batch size	—d	524288

Table 4: Flags which you could play with, but we don't recommend modifying

Description	Parameter	Default Value
val_loss_every, how often we evaluate val loss	—v	250
sample_every, how often we inference the model	—s	20000
hellaswag eval run?	—h	1

Table 5: Flags which you almost certainly won't need to change

1.6 Starter Strategies

These strategies should give you an idea of where to start and what we are expecting for your experimental design.

1. **Aggressive Greedy Algorithm**
   Experimental scaling: Number of training iterations
   Transfer hypothesis: No scaling
   In this strategy, you will find the learning rate that minimizes model loss after just a small number of epochs (50-100).

   Experimental overview: Start with an unreasonably high learning rate. Train the full-sized model for 10 steps with "warm-start" (the -u flag) disabled. If training does not diverge (in other words, the loss does not start going up) double the learning rate until it diverges. Can this learning rate $\alpha_{\text{divergent}}$. Then, train models for 10 steps with learning rates that are fractions of $\alpha_{\text{divergent}}$. You may want to perform several iterations to hone in on what the best scaling rate for your fractional values is.

   For the final training run, use the learning rate that performed best from these experiments.

2. **Early Stopping**
   Experimental scaling: Number of training iterations
   Transfer hypothesis: No scaling
   In this strategy, you will randomly sample learning rates and train for a fixed fraction of the total number of training epochs (for example, $\frac{1}{10}$). The learning rate which performs best from this cohort will be selected for the full training run.

   Experimental overview: Use grid or random search to perturb the starter learning rate (6e-4) in a large range above and below to generate 10 trial learning rates. Then, train the full-sized model for 100M tokens. For the final 2.5B token training run, select the learning rate that had the lowest validation loss from the 10 candidate models.
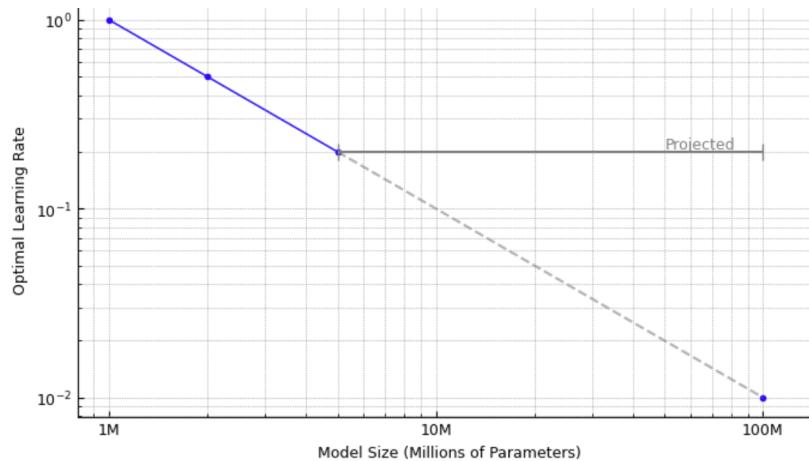
3. **Heuristic Model-Scaling Projection**
   Experimental scaling: Model size
   Transfer hypothesis: Functional scaling
   The "Heuristic Model-Scaling Projectionists" like to build mathematical models. In this strategy, you will train several tiny models of various scales to convergence. The goal is to fit a trend line to model scale vs optimal learning rate.

   "Heuristic Model-Scaling Projectionists" aim to build a plot like this:

Experimental overview: Define models with the following hyperparameters:

- Tiniest
    - Depth - 1
    - Channels - 64
    - Heads - 1
- Tinier
    - Depth - 2
    - Channels - 128
    - Heads - 2
- Tiny
    - Depth - 3
    - Channels - 192
    - Heads - 3

Use random search to train each model 10 times with a random hyperparameter alpha. Create a plot of model size vs. optimal learning rate, and extrapolate to the target model size of 124M paramters.

Select the learning rate projected to be optimal for a model of 124M parameters for your final training run.
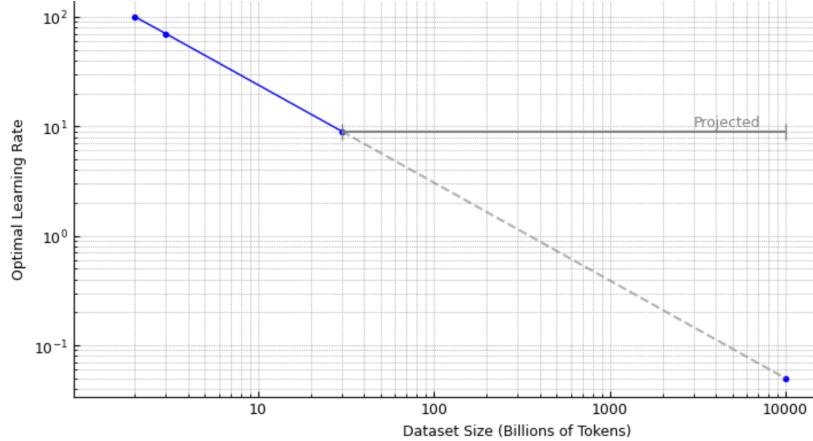
4. **Heuristic Data-Scaling Projection**
   Experimental scaling: Dataset size
   Transfer hypothesis: Functional scaling
   The "Heuristic Dataset-Scaling Projectionists" mostly see eye-to-eye with the "Heuristic Model-Scaling Projections", they just disagree on what should be scaled. These folks think that model scaling dynamics are more difficult to predict than data scaling dynamics. In this strategy, you will train several full-sized models on a single pass over datasets of various scales. The goal is to fit a trend line to dataset scale vs. optimal learning rate.

   "Heuristic Dataset-Scaling Projectionists" aim to build a plot like this:

Experimental overview: In this strategy, you will work with the following provided datasets:

Define a model with the same architecture as the final 124M parameter model.

- Depth 12

- Channels 768
- Heads 12

Now, define the following datasets as fractions of the real Fineweb4B dataset (this can be achieved using the -x flag):

- ∼26M tokens (50 steps with effective batch size 524288)
- ∼52M tokens (100 steps with effective batch size 524288)
- ∼105M tokens (200 steps with effective batch size 524288)

Use random search to train the model 10 times on each dataset with a random hyperparameter alpha. Create a plot of dataset size vs. optimal learning rate, and extrapolate to a dataset of 2.5B tokens.

Select the learning rate projected to be optimal for a model trained on a dataset of 2.5B tokens (4725 steps) for your final training run.

5. **Hyperband**
   Experimental scaling: Number of training iterations
   Transfer hypothesis: No scaling
   As discussed in lecture, Hyperband is a variant of early stopping that adds successive "halving" to evaluate more candidate models using the same computational budget.

   Experimental overview: find the HyperBand table below.

i	S=5		S=4		S=3		S=2		S=1		S=0	
	n_i	r_i	n_i	r_i	n_i	r_i	n_i	r_i	n_i	r_i	n_i	r_i
0	243	2	98	6	41	20	18	61	9	185	6	555
1	81	6	32	20	13	61	6	185	3	555		
2	27	20	10	61	4	185	2	555				
3	9	61	3	185	1	555						
5	3	185	1	555								
6	1	555										

   WARNING: This method requires evaluating a very large number of candidate models. You will therefore likely need to write some infrastructure code which can assist you to repeatedly schedule and save models.

   Reference: Hyperband: A Novel Bandit-Based Approach

6. **Hybrid**
   In this method, you will create a novel approach by mixing-and-matching several of the above approaches, scaling methods, and transfer methods. For example, you might combine HyperBand with Heuristic Model-Scaling Projection.

7. **To the Literature!**
   In this strategy, you will first perform a cursory literature review to identify the state-of-the-art method for single-shot hyperparameter tuning. This strategy will then be adopted and applied to the task in this homework.

   Here are some publications to get you started:

   - Language Models are Unsupervised Multitask Learners
   - Training Compute-Optimal Large Language Models
   - Tuning Large Neural Networks via Zero-Shot Hyperparameter Transfer

- Scaling Optimal LR Across Token Horizons

8. **None of the above**
   You are encouraged to apply your ingenuity and creativity to this problem and develop your own strategy for "zero-shot" hyperparameter tuning! We ask only that you do not exceed the outlined budget for your hyperparameter tuning strategy (in other words, please do not brute force this problem on a cluster if you have access to one).

1.7 Hints

1. It is **not** a good idea to exactly implement any of the starter strategies. You will need to make some tweaks and adjustments to get the strategy working. This section is only meant to set expectations and give a good place to start.

2. It is recommended to optimize for loss rather than HellaSwag accuracy

3. Make sure your hyperparameter search strategy has appropriate bounds. For example, if you are looking for a hyperparamter with optima at 10 and you search from 0.00001 to 1 you will never get the correct answer. Make sure not to make this mistake!

4. It is recommended to spend some small part of your budget getting familiar with the problem.

5. All hyperparameters in the final training run will be the default values show, except the minibatch size will be 32 and the max learning rate will be your selection.

1.8 FAQs

Q. Do really need to implement ALL of these experimental strategies?? That is too much work!
A. No. Please pick just ONE strategy. We gave a recommendation for a wide array of strategies because this topic is still an open research question and we hope that we can all learn together as a class which method works best.

Q. I don't really care for open-ended assignments! Can you please just tell me which strategy to use?
A. Sounds good! Select from strategies 1-4 (at random if necessary). These first four strategies are the most straightforward and are all fine choices.

Q. I don't really like structure! Can you please just let me do my own thing?
A. Sure! Please follow strategy "None of the above" which allows you the freedom to do as you please. We ask only that you make an honest effort at designing and implementing a strategy to find an optimal hyperparameter in a zero-shot fashion.

Q. I like a little but of structure, but please don't just throw me into the deep end! What should I do then?
A. Read over all of the strategies and see if any of them seem suitable to you. Pick whichever seems best to you, fill in the missing details needed and then (optionally) make some modifications to make a hybrid strategy or to improve the strategy.

Question 1: Running an Experiment *[15 points]*

In this question, you will be running an example experiment and analyzing the results. The goal of this is to familiarize yourself with the workflow, and maybe give you some ideas for how you will implement your own experimental design.

You will train a GPT-2 model that has 1 transformer layer, 64 channels, and 1 attention head for 1000 epochs, with 2 different learning rates, visualize the results, and analyze what you find.

To run the experiment, you will type this into the command line:

```
./train_gpt2cu \
    -o "question_1_experiment_1"  \
    -1d 1 \
    -1c 64 \
    -1h 1 \
    -l  0.01 \
    -u 70 \
    -x 20
```

Before we run the full experiment, we want to estimate how long the experiment will take and how much it will cost. We will therefore train it for just 20 steps and see how long it takes per step!

(a) *[1 points]* Run this experiment for 20 steps. The first (0th) step is an anomaly and is much slower due to memory loading. We will get a worst-case estimate from steps 10-19, just to be safe. Write down which step in the range step 10 to step 19 took the longest, and how long it took (in milliseconds).

(b) *[1 points]* Use this time-per-step value to estimate how long it will take to run the entire 1000 steps.

(c) *[1 points]* Now estimate the cost of this experiment. Using the formula below, obtain the estimated total runtime of the experiment in hours. Multiply this value by the cost of running an AWS g5.xlarge ($1.1066/hour at time of writing, but please round to $1.50/hour to be safe)

$$\texttt{Runtime (hours)} = \frac{\texttt{time per step (ms)} * \texttt{number of steps}}{3.6 \times 10^6 \texttt{ ms per hour}}$$

Now, run the experiment for the full 1000 steps using the command below. Note that the `-x` flag (which specifies the total number of steps we train for) has been changed. Once it has finished (set a timer using your prediction!), you will need to upload the results to your S3 bucket that you created during the AWS setup for this homework.

```
    ./train_gpt2cu \
        -o "question_1_experiment_1"  \
        -1d 1 \
        -1c 64 \
        -1h 1 \
        -l  0.01 \
        -u 70 \
        -x 1000
```

To upload the results to the bucket, run the following in your command line, changing the bucket name to the one you created.

```
    python upload_to_s3.py "your_s3_bucket_name" \
                           "question_1_experiment_1" \
                           "question_1_experiments"
```

The first argument is your bucket name. The second argument is the local path to the output files, which doubles as the experiment name (note that this is exactly the same as what we passed into the `-o` flag when running the experiment). The third argument specifies the 'run cluster'. Think of it as naming the collection of experiments to which this specific experiment belongs.

Once you have completed this. Do another run for 1000 steps, but this time change the learning rate using the `-l` flag to 0.001 from 0.01, and also change the `-o` flag to `"question_1_experiment_2"`. Once that has finished, upload those results to your S3 bucket by typing the following into your command line. Notice that we changed the second argument accordingly.

```
    python upload_to_s3.py "your_s3_bucket_name" \
                           "question_1_experiment_2" \
                           "question_1_experiments"
```
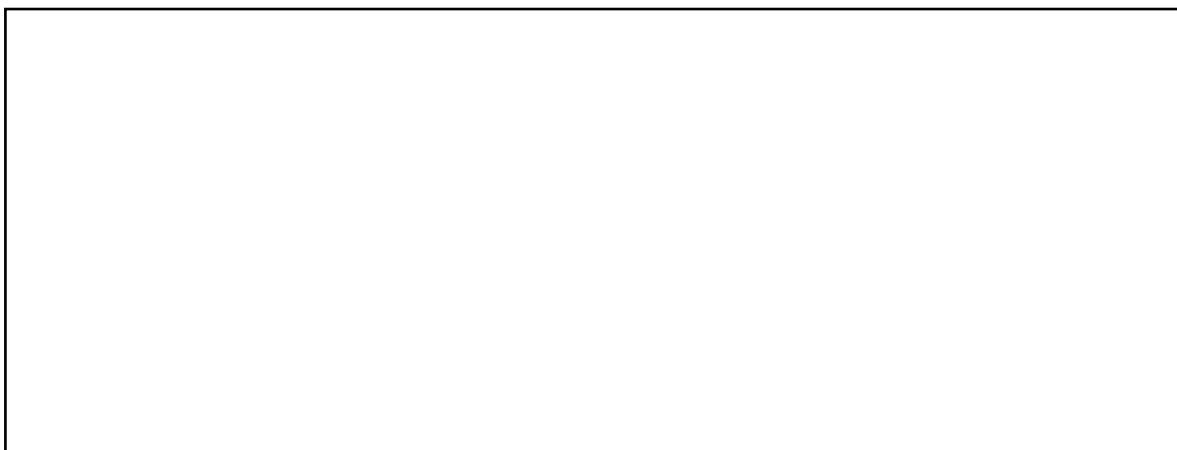
After having uploaded both experiments to your S3 bucket, you can now shut down the instance by running `sudo shutdown -h now`. This will close the instance and prevent you from being charged (Note that you will still be charged for the volume, but this amount is negligible compared to the instance.)

Now that you have uploaded your logs to the S3 bucket, it's time to visualize and interpret your results! Download and open the `vislog.ipynb` notebook, and fill in your AWS access key and AWS secret key that you got when creating the IAM user. Also fill in your AWS bucket. Then, follow the instructions in the notebook to make to make the plots and complete the following questions.
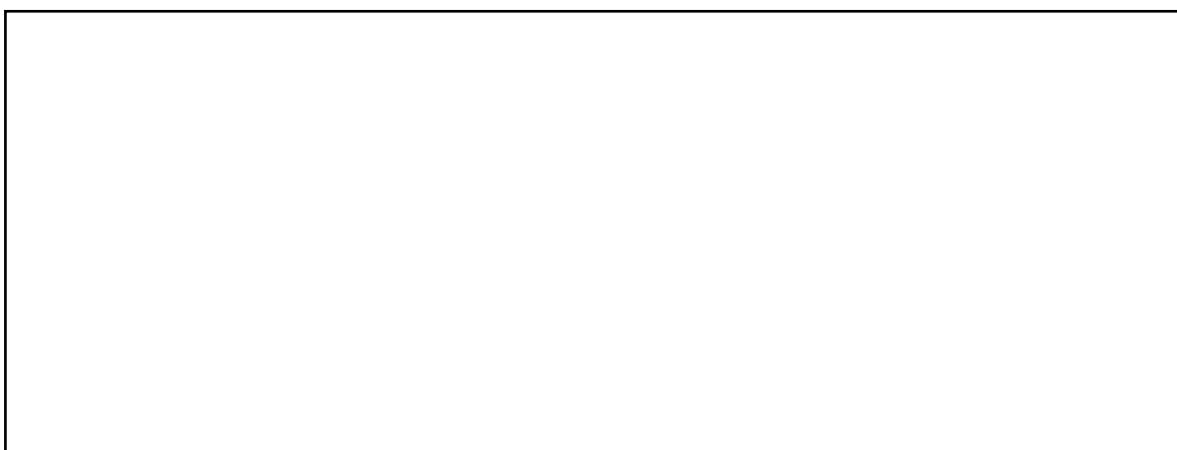
(d) *[2 points]* Make one plot that contains the training and validation loss for the first experiment you ran.

(e) *[2 points]* Make one plot that contains the training and validation loss for the second experiment you ran.

(f) *[2 points]* Make one plot that contains the training loss for both experiments, and displays the minimum training loss for each experiment on the graph.

(g) *[2 points]* Make one plot that contains the validation loss for both experiments, and displays the minimum validation loss for each experiment on the graph.

(h) *[2 points]* List how many parameters the model has, and the minimum training and validation loss for both learning rates.

(i) *[2 points]* Based on the results of these experiments, what other learning rates would you test if you had to find the optimal learning rate for this specific model and hyperparameters? Would you try larger or smaller learning rates?

Congratulations, you just completed an experiment! If you don't plan to continue working on your own experiments right now, please shut down your instance by running `sudo shutdown -h now` so you don't get charged!

Note: We recommend writing bash or shell scripts to automatically run experiments sequentially. This is not required to complete this homework, but it makes running experiments much more cost efficient and easier.

Question 2: Tuning a small model *[10 points]*

Consider the following plots of how the optimal learning rate scales with respect to model width (note that this is an multi-layer perceptron trained for 20 epochs on CIFAR-10).


Figure 2: Source: https://arxiv.org/pdf/2203.03466
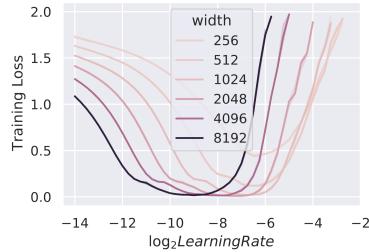
Note how the optimal learning rate for a given model width forms a nice convex U-shaped surface when plotted in log space. Emperically, we have observed that transformer-class modles exhibit the same behavior.

In this question, you will be using this knowledge to find the optimal max learing rate for a small compute-optimal model with 10% learning rate warmup.

(a) *[1 points]* Compute-optimal large language models should be trained on roughly 20 tokens of data per parameter [5]. Using this knowledge, how many tokens should be used to train a compute optimal "tiniest" model?

(b) *[1 points]* "llm.c" uses an effective batch size of 524,288. This means that gradients in mini-batches are accumulated until 524,288 tokens have been processed. Only at this time does the optimizer take a step.

Given this information, how many steps (set using the -x flag) should a compute-optimal "tiniest" model be trained for?
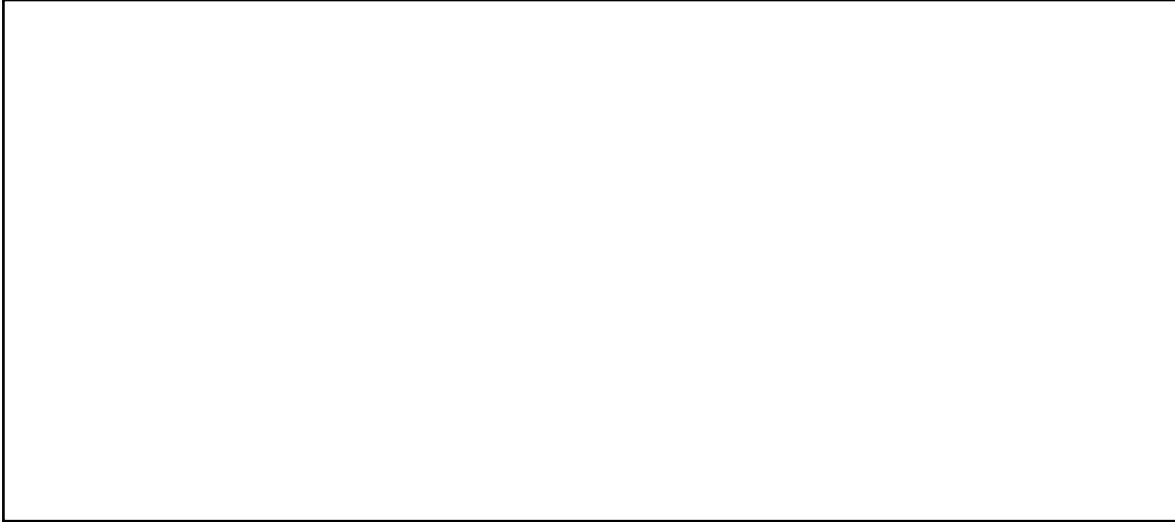
(c) *[6 points]* Find the optimal learning rate for a compute-optimal "tiniest" model trained using the flags below.

```
./train_gpt2cu \
    -o "question_2"  \
    -1d 1 \
    -1c 64 \
    -1h 1 \
    -u 13 \
    -x 128 \
    -l  <YOUR LEARNING RATE HERE>
```
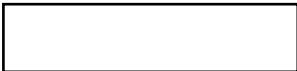
Hint: This should be a U-shaped curve
Hint: Make sure to set the bounds of your search correctly. If the optimum is (for example) at 1e-8 and you are searching from 1e-5 to 1e1, you will never find the optimum.

16

Submit your log-learning rate vs training loss plot below.
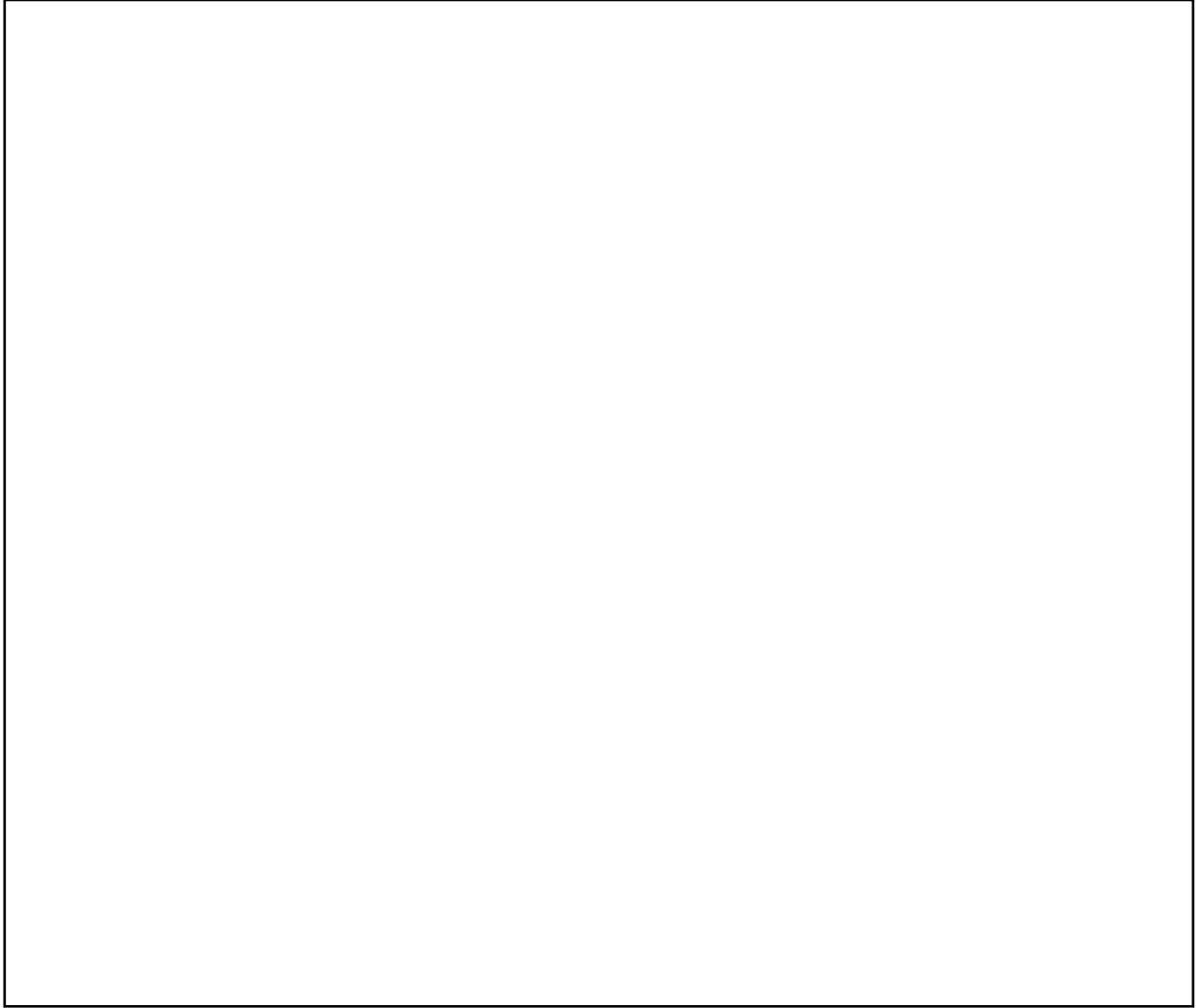
(d) *[2 points]* What is the optimal learning rate that you found?

Question 3: Experimental overview

[10 points] Now, design your own experiments (10 mandatory experiments, more optional) for solving the problem of zero-shot hyperparameter tuning. Read over the starter strategies in section 2.6. Fill in missing details and make modifications as needed. Describe in words your experimental overview. Here are some (optional) questions for you to consider:

- What strategies will you test?

- What outcome do you expect to see from your experiments?

- How will you evaluate if your methodology is working?

- How will you know if you need to revise your strategy and iterate again?

- What (if any) kind of iteration will you perform to adjust your strategy if necessary?

Question 4: Fill in your Experimental Table

[10 points points] Now that you have planned your experiments, please proceed to run your experiments and complete filling in the first 10 rows of the table below. 10 rows are required, but you may use the remainder as needed. Make sure not to mix up your items. Please be sure to fill in the validation loss and HellaSwag accuracy.

Question 5: Select your Final Hyperparameter Value

(a) *[2 points]* Please report your total expenditure in dollars on this programming part 1 assignment.

(b) *[2 points]* On the basis of your experiments above, please select the learning rate you plan to use for the final full training run.

```
┌─────────────────┐
│                 │
│                 │
└─────────────────┘
```

(c) *[1 points]* If you elected not to select the learning rate with the best performance in your experiments, please briefly motivate your selection below. For example, explain your transfer hypothesis.

```
┌─────────────────────────────────────────────────────────────────────────┐
│                                                                         │
│                                                                         │
│                                                                         │
│                                                                         │
└─────────────────────────────────────────────────────────────────────────┘
```

Section 6: Accuracy

[10 points points] Finally, a small component of your grade will depend on how close your selected learning rate is to the true global optimum. Any value within a "reasonable" distance will receive full credit. The definition of "reasonable" will be determined by your performance relative to past and current students.

Experiment number	Max learning rate	Training steps	Depth	Channels	Heads	Ramp up? (Y/N)	Learning rate decay target	Run time	Approx. cost	Final val loss	Final HellaSwag accuracy
1											
2											
3											
4											
5											
6											
7											
8											
9											
10											
11 (optional)											
12 (optional)											
13 (optional)											
14 (optional)											
15 (optional)											
16 (optional)											
17 (optional)											
18 (optional)											
19 (optional)											
20 (optional)											
21 (optional)											
22 (optional)											
23 (optional)											
24 (optional)											
25 (optional)											
26 (optional)											
27 (optional)											
28 (optional)											
29 (optional)											
30 (optional)											

2 Programming Part 2 [20 points]

From URLs to Logits Part 2: Pre-processing Training Data

2.1 Introduction

Before we can train a model on the dataset we curated in homework 4, we will need to pre-process the data and wrangle it into the format needed by our LLM. Many of these steps should look familiar, building upon concepts that you have mastered in prior homeworks:

1. Document deduplication using MinHash LSH

2. Distributed processing using Map-Reduce systems

Unfortunately, using large datasets in a practical application does often come with some negative aspects:

1. Long running times. Expect many steps, each of which can take up to 12 hours. We will do everything we can to ameliorate this (providing step-by-step video tutorials and slide decks with instructions) but it cannot be avoided.

2. Expensive computation. Our problems are scaled-down but still have potential to cause very large AWS bills if you are not careful.

2.2 Note on Homeworks with Continuity

This homework section builds heavily on previous homeworks. If you were unable to complete your dataset curation in Homework 4, please email the course staff (jrast@andrew.cmu.edu) or make a post on Piazza. We will give you the resources necessary to complete each section independently, but may levy a small penalty (-5 or -10 points to the homework grade) to give these resources if you make an unreasonable request. This penalty is to discourage anyone from trying to take a shortcut.

2.3 Instructions

At a high level, you will need to do the following. For detailed instructions, please see the accompanying video tutorials and slide deck posted on Piazza.

1. Spin up an EMR cluster:

   - emr-7.7.0
   - Hadoop 3.4.0, Spark 3.5.3, JupyterHub 1.5.0
   - 1 primary + 7 core r5.xlarge instances
   - 100 GB EBS root volume
   - 200 GB EBS core volume (click "Actions" → "Edit EBS Volumes")
   - Auto-terminate after 10 minutes idle
   - Bootstrap script:

     ```
     #!/bin/bash
     sudo python3 -m pip install --ignore-installed \
     'urllib3<2.0' requests beautifulsoup4 numpy pandas \
     tiktoken boto3 warcio
     ```

2. Upload the Homework 5 Coding 2 Jupyter notebook handout

3. Complete the TODOs for a small dataset of 1k documents.

4. Download the deduped 1k dataset and submit to Gradescope for autograding.

5. Complete the TODOs for a slightly larger dataset of 10k documents.

6. Download the deduped 10k dataset and submit to Gradescope for autograding.

7. (Recommended, but optional): Complete the TODOs for 500k documents. Convert the Jupyter notebook to Python script.

8. Complete the TODOs for the full run of 5 million documents. Convert the Jupyter notebook to Python script.

9. Upload completed Python files to to S3.

10. Connect to EMR primary node via SSH

11. Start screen

12. Begin preprocessing your data. You should run the following command:

```
spark-submit --deploy-mode client --master yarn \
--conf spark.driver.memory=20g \
--conf spark.driver.maxResultSize=15g \
--num-executors 7 --executor-memory 20GB \
s3://[path/to/preprocessing.py]
```

2.4 Getting Started

1. **Getting lab files**
   You can obtain the handout 'hw5_preprocessing.ipynb' for this programming questions in .zip file.

2. **Getting dataset**

   (a) The 1k dataset can be found in the public S3 bucket "10605-f25-hw5-subset-1000"
   (b) The 10k dataset can be found in the public S3 bucket "10605-f25-hw5-subset-10000"

2.5 Deliverables

For this section of the homework, you will be expected to submit the following:

1. A completed written handout

2. A completed 'hw5_preprocessing.ipynb', uploaded to Gradescope.

3. A list of the deduplicated 1k dataset, in .snappy.parquet format. This list should be uploaded to Gradescope for autograding.

4. A list of the deduplicated 10k dataset, in .snappy.parquet format. This list should be uploaded to Gradescope for autograding.

Question 1: 1K Dataset Preprocessing *[8 points]*

When faced with a large data problem, it is a good idea to begin with a small subset of the data and progressively increase the dataset size. This is so that you can begin developing, debugging, and optimizing your code on small, cheap compute clusters and avoid the large costs and long development cycles that accompany big dataset.

We have provided you with a small set of 1,000 English-language documents. Your task will be to deduplicate the documents and upload a list of documents to Gradescope for autograding. Your score will be determined by your ability to reach a target precision and recall.

Question 2: 10K Dataset Preprocessing *[8 points]*
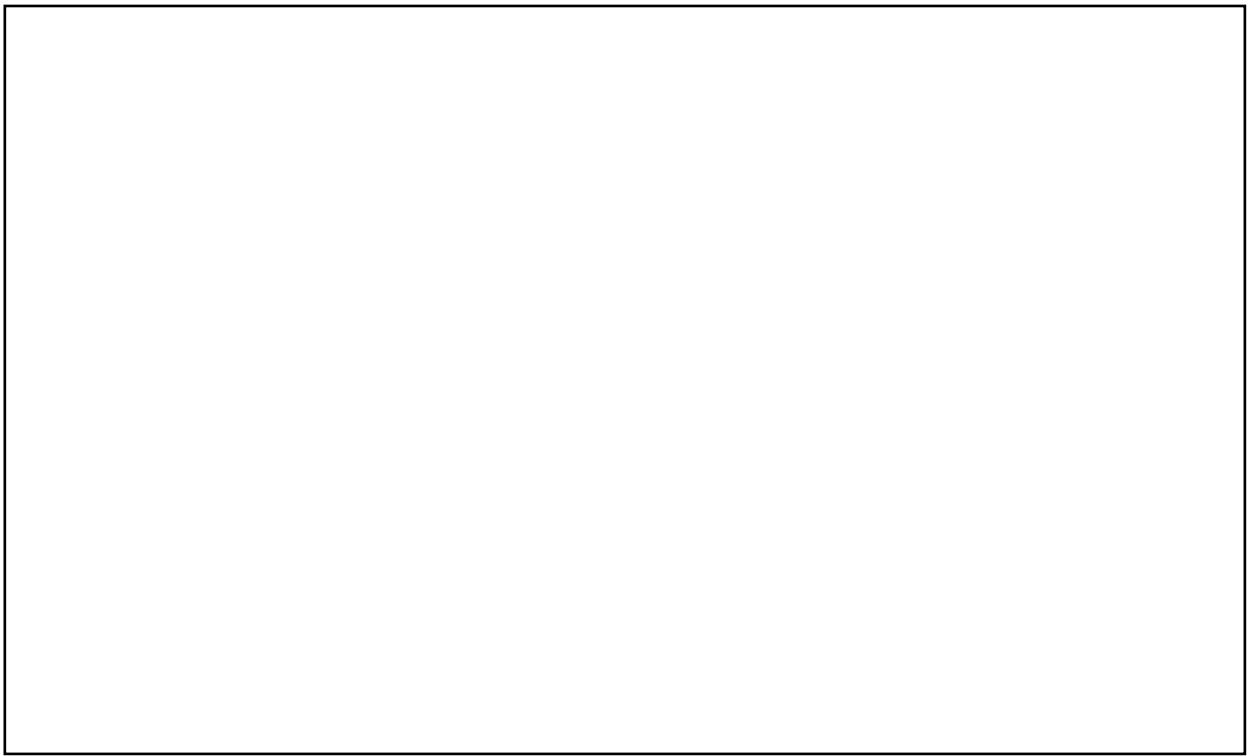
Now complete the same process for a dataset of 10,000 English-language documents. You will need to make sure your methodology scales well.

Question 3: Large Dataset Preprocessing *[4 points]*

Now, scale your code up to run on the dataset the your curated in Homework 4. Please upload a screenshot of the S3 bucket containing your tokenized dataset. Make sure your AndrewID is clearly visible.

HINTs:

- Make sure to remove any intermediate actions, such as .take(1) or .count(). These actions can easily cause repeated computation.

- Do not call coalesce. You will crash your cluster.

- You may want to test on a subset of 500,000 documents to make sure your code is scalable and bug-free at scale. This should take less than 1 hour.

- Use screen. The final deduplication run may take up to 12 hours.

- If you are unable to complete the large dataset tokenization, the course staff can provide you with a dataset so you can complete other parts of the homework. See "Note on Homeworks with Continuity" elsewhere in the homework.

3 Programming Part 3 [10 Points]

From URLs to Logits Part 3: Branch Yesterday, Train Today, Merge Tomorrow

3.1 Introduction

Now that you have finished hyperparameter tuning and dataset preprocessing, you will be able to pre-train your own (small-scale) foundation model! In this homework section, your objective is to pre-train a 124M parameter GPT-2 Small model on 2.5B tokens of data for a single epoch of 4725 training steps.

However, Branch-Train-Merge requires models initialized from a common "seed model" trained on approximately 50% of the total training data. In order to facilitate ensembling the entire class's models, we have provided all of you with a shared initialization. This "seed model" has been trained on approx 1.25B tokens of FineWeb data. You will be responsible for completing the model training on your custom dataset!

3.2 Note on Homeworks with Continuity

This homework section builds heavily on previous homeworks. If you were unable to complete your dataset curation in Homework 4 or unable to complete the homework preprocessing in Homework 5 Programming Part 1, please email the course staff (jrast@andrew.cmu.edu) or make a post on Piazza. We will give you the resources necessary to complete each section independently, but may levy a small penalty (-5 or -10 points to the homework grade) to give these resources if you make an unreasonable request. This penalty is to discourage anyone from trying to take a shortcut.

3.3 Instructions

3.3.1 AWS EC2 Details

In this section you will once again be using an g5.xlarge instance. However, there is a new AMI for you to use with a halfway trained model. See the getting started section for details!

3.3.2 Reformatting Parquet Files For GPT2

Believe it or not, there is still one last pre-processing step to get your dataset into the format required for training an LLM.

1. Upload the `parquet_processor_v2.py` script to your instance.

2. Modify the script to include your data location in your S3 bucket.

3. Run `parquet_processor_v2.py`. Expected runtime: approximately 1 hour.

3.3.3 Downloading Files from an S3 Bucket to an AWS Instance

You can use the following command to download a directory from an S3 bucket. Make sure you give permissions to access the S3 bucket using the command "AWS Configure" first.

```
aws s3 cp "s3://<your_s3_bucket_name>/path/to/directory" "<directory>" --recursive
```

3.3.4 Final Pretraining Run

Use the following command to start your full pre-training run. Make sure to use `tmux` or `screen` so that your training run does not get interrupted part way through!

```
./train_gpt2cu \
    -o "logGPT"  \
    -1d 12 \
    -1c 768 \
    -1h 12 \
    -b 32 \
    -y 1 \
    -e "path/to/seed_model.bin" \
    -ss 2365 \
    -l <YOUR LEARNING RATE HERE>
```

WARNING: The total cost of training this model is approx. $10 and you may not have sufficient budget to train it twice if something goes wrong. Please make sure to do the following:

1. Use `tmux` or `screen` so the process is not killed if your ssh connection is closed.

2. Do not terminate your g5.xlarge instance until you have triple-checked that your model weights and training log are stored somewhere safe.

3. Babysit your process. Check in $\sim$ 30 minutes into the training run to make sure everything is going smoothly

4. Do not leave your instance running after training is finished. Don't start a run that will complete at 3am. Don't forget that your model is training and leave it running for several days.

3.3.5 Downloading log files

After completing your model pretraining, you can run the following command to copy your log files to s3:

```
python upload_to_s3.py "<your_s3_bucket_name>" \
                        "logGPT" \
                        "full_gpt"
```

Then use the "`vislog.ipynb`" file to create plots of your training run.

3.3.6 Downloading a file from an AWS Instance

You will need to submit you model weights. This will require downloading them from AWS. You can use one of the following methods to do so.

Method 1: Upload to S3 (recommended)

1. Run the following command to upload your model weights to S3:

```
aws s3 cp "logGPT/" "s3://<your_s3_bucket_name>" --recursive
```

2. Download the file from S3 for submission:

Note: this method can be automated. You can use this method in a bash script to automatically upload weights to s3. Add `sudo shutdown -h now` to automatically stop your instance once training and upload are complete.

Method 2: Use `scp`
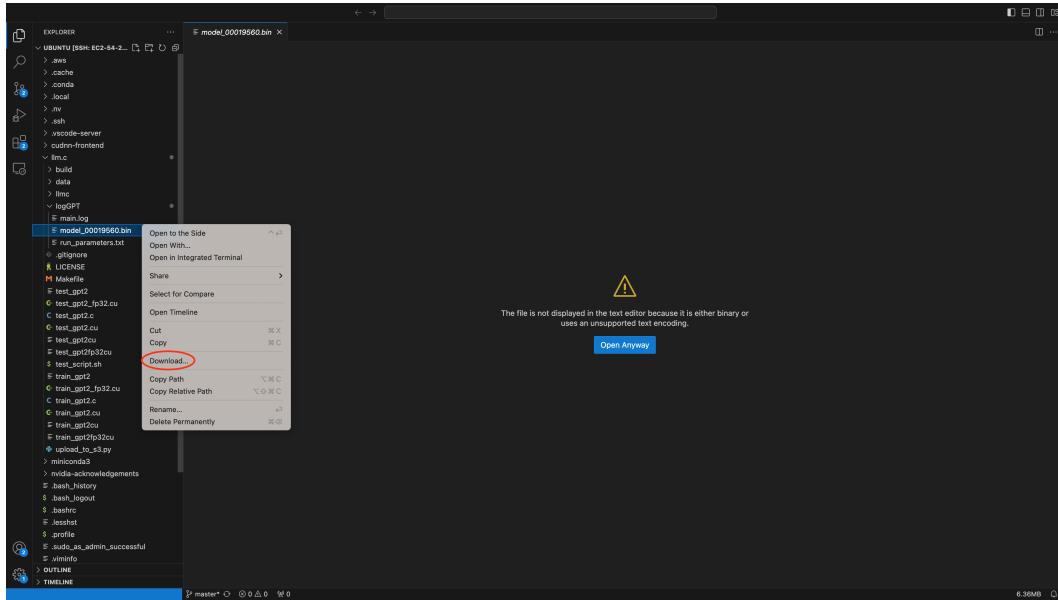
1. Run the following command on your local computer to download the weights directly from AWS to your computer:

```
scp -i <key pem file> -r \
ubuntu@<AWS instance public DNS>:/home/ubuntu/llm.c/logGPT/ .
```

   Note: the section of this command `ubuntu@<AWS instance public DNS>` should be identical to a portion of the `ssh` command used to connect to your instance.
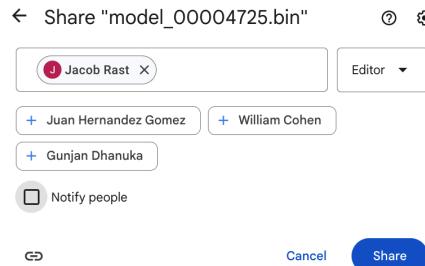
Method 3: Use VS Code

If you are connected to your AWS g5.xlarge instance using VS Code, you can simply download the weights by right-clicking and selecting download:

3.3.7 Submitting your model weights

Upload your model weights to Google Drive and share them with jrast@andrew.cmu.edu.

3.4 Getting Started

The following is a high level overview of what you will need to do to setup your AWS instance and begin your pretraining run. For a much more detailed guide, see the recitation materials.

- Create an IAM role and user (save your access key and secret key somewhere safe!). This is required to save your training logs to an S3 bucket.

- Create a g5.xlarge instance using 'ami-0adb6bdc4b8d3de63' and providing the IAM role.

- Download your curated dataset to the instance from your S3 bucket

- Run your final full pretraing run

- Download the model weights and training log

- Visualize the training run and share the model weights with the course staff

28
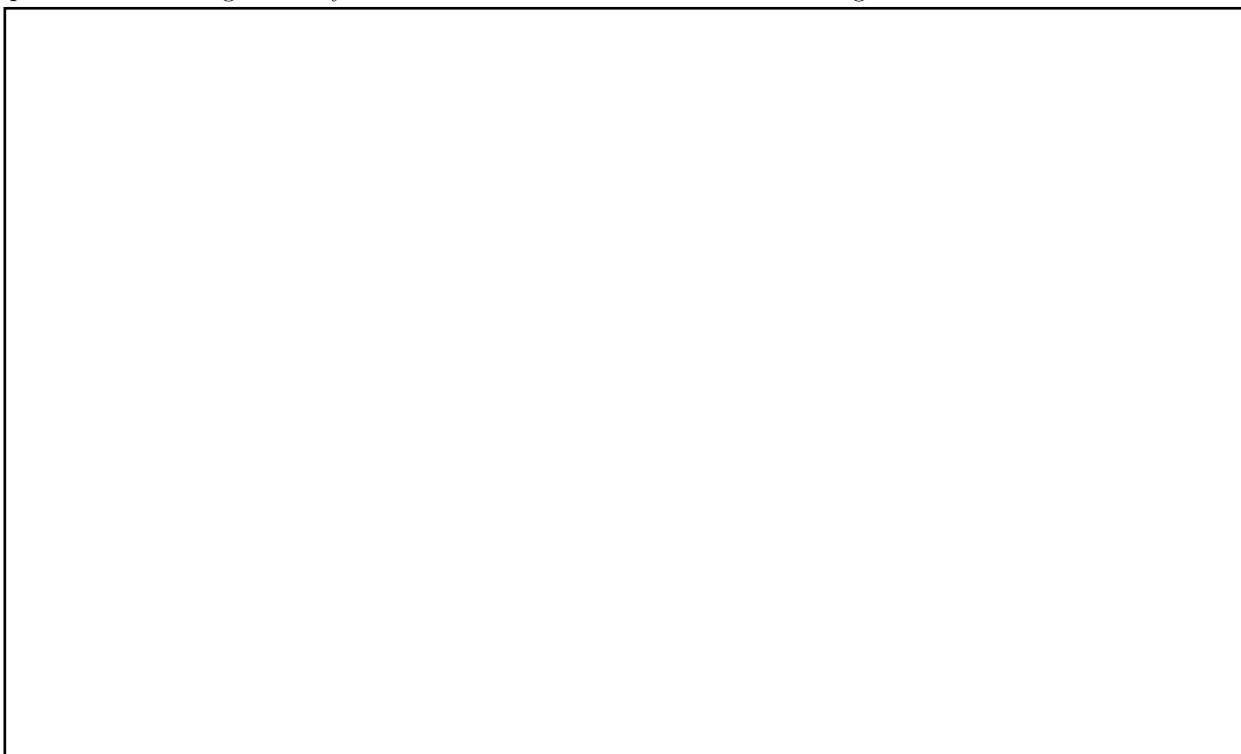
3.5 Deliverables

For this programming section you will have the following deliverables:

1. Share your model weights with jrast@andrew.cmu.edu. **Please make sure to submit your final weights via Google Drive or you will receive a score of 0 for this section of the homework**.

2. Completed the written handout

3. Completed assignment "Homework 5 Programming Part 3" on Gradescope

Question 1: Final Loss Plot

[10 points] Please use the helper file "vislog.ipynb" in order to visualize the training and validation loss on one plot and HellaSwag accuracy on another for the final GPT-Small training run.

4 Programming Part 4 (optional)

From URLs to Logits Part 4: Inference *[0 points]*

Now that you have finished training your language model, you can finally play with it! Your models are quite small, have not been fine-tuned for chatting, and are not competitive against a billion or trillion parameter language model. However, you can use the "inference_gpt2" binary in order to have your model autocomplete a prompt. You can see an example below:

```
(base) ubuntu@ip-172-31-27-23:~/llm.c/HW5_Code$ ./inference_gpt2cu -e "/home/ubuntu/llm.c/HW5_Code/model_00004725.bin" -p "The GitHub project llm.c is a" -it "/home/ubuntu/f24/llm.c/data/gpt2_tokenizer.bin"
Multi-GPU support is disabled. Using a single GPU.
[System]
Device 0: NVIDIA A10G
Loading model from /home/ubuntu/llm.c/HW5_Code/model_00004725.bin
Prompt: "The GitHub project llm.c is a"
Generating 256 tokens...

| max_sequence_length T | 1024        |
| vocab_size V          | 50257       |
| padded_vocab_size Vp  | 50304       |
| num_layers L          | 12          |
| num_heads NH          | 12          |
| channels C            | 768         |
| num_parameters        | 124475904   |
+-----------------------+-------------------------------------------------+
allocating 331 MiB for activations
Tokenizing prompt...
Prompt tokens: 464 21722 1628 32660 76 13 66 318 257

Generated text:
---
The GitHub project llm.c is a
---
 free and open source latin wordpress project with its own font and its own author numbers. python is a kick ass txt file loaded with features.<|endoftext|>
[Generation stoped: EOT token]

---
```

Not bad for a model trained pretrained on a single GPU with just a few bucks of AWS credits :P

5   Collaboration Questions

1. (a) Did you receive any help whatsoever from anyone in solving this assignment?

   (b) If you answered 'yes', give full details (e.g. "Jane Doe explained to me what is asked in Question 3.4")

2. (a) Did you give any help whatsoever to anyone in solving this assignment?

   (b) If you answered 'yes', give full details (e.g. "I pointed Joe Smith to section 2.3 since he didn't know how to proceed with Question 2")

3. (a) Did you find or come across code that implements any part of this assignment?

   (b) If you answered 'yes', give full details (book & page, URL & location within the page, etc.).