

HOMework 1

GENERATIVE MODELS OF TEXT *

10-423/623/723 GENERATIVE AI
<http://423.mlcourse.org>

OUT: Jan. 27, 2026
DUE: Feb. 09, 2026
TAs: Natalie, Gopal, Dhruv

Instructions

- **Collaboration Policy:** Please read the collaboration policy in the syllabus.
- **Late Submission Policy:** See the late submission policy in the syllabus.
- **Submitting your work:** You will use Gradescope to submit answers to all questions and code.
 - **Written:** You will submit your completed homework as a PDF to Gradescope. Please use the provided template. Submissions can be handwritten, but must be clearly legible; otherwise, you will not be awarded marks. Alternatively, submissions can be written in $\text{\LaTeX}$. Each answer should be within the box provided. If you do not follow the template, your assignment may not be graded correctly by our AI assisted grader and there will be a **2% penalty** (e.g., if the homework is out of 100 points, 2 points will be deducted from your final score).
 - **$\text{\LaTeX}$ Source:**
 - **Programming:** You will submit your code for programming questions to Gradescope. We will examine your code by hand and may award marks for its submission.
- **Materials:** The data that you will need in order to complete this assignment is posted along with the writeup and template on the course website.

Question	Points
$\text{\LaTeX}$ Template Alignment	0
Recurrent Neural Network (RNN) Language Models	7
Transformer Language Models	19
Sliding Window Attention	11
Programming: RoPE and GQA	22
Code Upload	1
Collaboration Questions	2
Total:	62

*Compiled on Tuesday 27th January, 2026 at 11:12

1 $\text{\LaTeX}$ Template Alignment (0 points)

1.1. (0 points) **Select one:** Did you use $\text{\LaTeX}$ for the entire written portion of this homework?

☐ Yes

☐ No

1.2. (0 points) **Select one:** I have ensured that my final submission is aligned with the original template given to me in the handout file and that I haven't deleted or resized any items or made any other modifications which will result in a misaligned template. I understand that incorrectly responding yes to this question will result in a penalty equivalent to 2% of the points on this assignment.

Note: Failing to answer this question will not exempt you from the 2% misalignment penalty.

☐ Yes

2 Recurrent Neural Network (RNN) Language Models (7 points)

- 2.1. (3 points) **Numerical answer:** Consider an RNN (Elman Network) that takes inputs $\mathbf{x}_t \in \{0, 1\}^2$, has hidden vectors $\mathbf{h}_t \in \mathbb{R}^2$, and output units $y_t \in \mathbb{R}$ for all $t \in \{1, \dots, T\}$. Assume the recurrence is given by:

$$\begin{aligned} h_t &= \text{slide}(W_{hh}h_{t-1} + W_{hx}x_t + b_h) \\ y_t &= \text{slide}(W_{yh}h_t + b_y) \end{aligned}$$

where $\text{slide}(a) = \min(1, \max(0, a))$ is the activation function. Note that when the slide function is applied to a vector, it is applied to each element of the vector individually.

Let $W_{hh} \in \mathbb{R}^{2 \times 2}$ be defined as follows:

$$W_{hh} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

Define parameters $W_{hx} \in \mathbb{R}^{2 \times 2}$, $W_{yh} \in \mathbb{R}^{1 \times 2}$, $b_h \in \mathbb{R}^2$, $b_y \in \mathbb{R}$ to satisfy the following condition: $y_t = 1$ if $\exists r, s \leq t$ such that $x_{r,0} = 1$ and $x_{s,1} = 1$ and $y_t = 0$ otherwise. Assume $h_0 = [0, 0]^T$.

W_{hx}

b_h

W_{yh}

b_y

2.2. An autoregressive language model defines a probability distribution over sequences $\mathbf{x}_{1:T}$ of the form: $p(\mathbf{x}_{1:T}) = \prod_{t=1}^T p(x_t | x_1, \dots, x_{t-1})$.

2.2.a. (2 points) **Short answer:** Suppose we are given an input $\mathbf{x}_{1:T}$ and we define a bidirectional RNN of the following form:

$$\begin{aligned} f_t &= \sigma(W_{ff}f_{t-1} + W_{fx}x_t + b_f), \quad \forall t \in \{1, \dots, T\} \\ g_t &= \sigma(W_{gg}g_{t+1} + W_{gx}x_t + b_g), \quad \forall t \in \{1, \dots, T-1\} \\ h_t &= \sigma(W_{hf}f_t + W_{hg}g_t + b_h), \quad \forall t \in \{1, \dots, T\} \end{aligned}$$

(Notice that f_t builds up context from the left, g_t builds up context from the right, and h_t combines the two.) Can we define an autoregressive language model of the form $p(\mathbf{x}_{1:T}) = \prod_{t=1}^T p(x_t | h_{t-1})$?

If so, how can we compute $p(x_t | \text{BiRNN}(\mathbf{x}_{1:t-1}))$ using this RNN? If we can't define an autoregressive LM, why not? Assume that **no** weight matrix can be set to all zeros.

2.2.b. (2 points) **Short answer:** Suppose $\text{BiRNN}(\mathbf{x}_{1:t-1})$ computes a bidirectional RNN on the subsequence $\mathbf{x}_{1:t-1}$ and then returns h_{t-1} . Can we define an autoregressive language model of the form $p(\mathbf{x}_{1:T}) = \prod_{t=1}^T p(x_t | \text{BiRNN}(\mathbf{x}_{1:t-1}))$?

If so, how can we compute $p(x_t | \text{BiRNN}(\mathbf{x}_{1:t-1}))$ using this RNN? If we can't define an autoregressive LM, why not?

(Notice that here we are only looking at the part of the bidirectional RNN which goes from left to right and looks at all the words before t .)

3 Transformer Language Models (19 points)

3.1. Transformers use scaled-dot-product attention:

$$s_{t,j} = \mathbf{k}_j^T \mathbf{q}_t / \sqrt{|\mathbf{k}|}, \forall j, t$$

$$\mathbf{a}_t = \text{softmax}(\mathbf{s}_t), \forall t$$

where the values, queries, and keys are respectively given by: $\mathbf{v}_j = \mathbf{W}_v^T \mathbf{x}_j$, $\mathbf{q}_j = \mathbf{W}_q^T \mathbf{x}_j$, and $\mathbf{k}_j = \mathbf{W}_k^T \mathbf{x}_j$ for all j and $\mathbf{v}_j, \mathbf{q}_j, \mathbf{k}_j \in \mathbb{R}^{d_k}$.

3.1.a. (2 points) **Short answer:** In the attention mechanism, queries, keys, and values are all derived from the same input representations. Conceptually, why do we need all three, and how does the dot product of queries and keys determine what information from the values gets passed forward?

3.1.b. (4 points) **Numerical answer:** Consider the 3-token sentence "*birds fly away*". Let each token have a one-hot input embedding in $\mathbb{R}^3$:

$$X = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad (\text{rows correspond to "birds", "fly", "away"}).$$

We use the following projection matrices (single head with $d_k = d_v = 2$, sequence length $T = 3$):

$$\mathbf{W}_q = \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 1 & 1 \end{bmatrix}, \quad \mathbf{W}_k = \begin{bmatrix} 1 & 0 \\ 1 & 1 \\ 0 & 1 \end{bmatrix}, \quad \mathbf{W}_v = \begin{bmatrix} 2 & 0 \\ 0 & 3 \\ 1 & 1 \end{bmatrix}.$$

Compute one attention pass (single head), following the steps on the next page.

(i) Compute the values of the projections:

$$\mathbf{Q} = \mathbf{X}\mathbf{W}_q, \quad \mathbf{K} = \mathbf{X}\mathbf{W}_k, \quad \mathbf{V} = \mathbf{X}\mathbf{W}_v.$$

Q	K	V

(ii) Compute the score matrix $\mathbf{S} = \frac{\mathbf{Q}\mathbf{K}^\top}{\sqrt{d_k}}$ and the attention matrix $\mathbf{A} = \text{softmax}(\mathbf{S})$ computed *row-wise*. (Report your answer rounded to 3 decimal places.)

For this subpart only, use the following fixed matrices (do not use part (i)):

$$d_k = 2, \quad Q = \begin{bmatrix} 2 & 0 \\ 0 & 1 \\ 1 & 2 \end{bmatrix}, \quad K = \begin{bmatrix} 1 & 0 \\ 2 & 1 \\ 1 & 1 \end{bmatrix}.$$

S	A

(iii) Compute the output $\mathbf{X}' = \mathbf{A}\mathbf{V}$. (Report your answer rounded to 3 decimal places.)

For this subpart only, use the following fixed matrices (do not use your answers from earlier parts):

$$A = \begin{bmatrix} 0.200 & 0.500 & 0.300 \\ 0.100 & 0.700 & 0.200 \\ 0.600 & 0.200 & 0.200 \end{bmatrix}, \quad V = \begin{bmatrix} 1 & 2 \\ 0 & 1 \\ 3 & 1 \end{bmatrix}.$$

X'

(iv) In 1–2 sentences, state what each row of X' represents.

For this subpart only, use the following fixed attention matrix (do not use your answer from (ii)):

$$A = \begin{bmatrix} 0.200 & 0.500 & 0.300 \\ 0.100 & 0.700 & 0.200 \\ 0.600 & 0.200 & 0.200 \end{bmatrix}.$$

(Hint: The token “away” corresponds to row 3. With this A , “away” assigns its highest attention to token “birds”.)

3.2. As mentioned in 3.1., Transformers use scaled-dot product attention.

3.2.a. (2 points) **Short answer:** Multiplicative attention instead defines the attention weights as:

$$\tilde{s}_{t,j} = \mathbf{k}_j^T \mathbf{W}_s \mathbf{q}_t / \sqrt{|\mathbf{k}|}, \forall j, t$$

$$\tilde{\mathbf{a}}_t = \text{softmax}(\tilde{\mathbf{s}}_t), \forall t$$

where $\mathbf{W}_s \in \mathbb{R}^{d_k \times d_k}$ is a learned parameter matrix. Could a Transformer with multiplicative attention learn a function that a Transformer with scaled dot product attention cannot?

Note: Two functions are equivalent if they produce the same output for every input in the domain.

3.2.b. (2 points) **Short answer:** Concatenated attention defines the attention weights as:

$$\hat{s}_{t,j} = \mathbf{w}_s^T [\mathbf{k}_j; \mathbf{q}_t], \forall j, t$$

$$\hat{\mathbf{a}}_t = \text{softmax}(\hat{\mathbf{s}}_t), \forall t$$

where $\mathbf{w}_s \in \mathbb{R}^{2d_k}$ is a parameter vector, and $[\mathbf{a}; \mathbf{b}]$ is the concatenation of vectors $\mathbf{a}$ and $\mathbf{b}$. Do there exist parameters, that can be learned, $\mathbf{w}_s$ such that $\hat{s}_{t,j}$ will approximately equal the angle θ between the two vectors $\mathbf{k}_j, \mathbf{q}_t$, or to $\cos(\theta)$? (Briefly justify your answer—a formal proof is not required.)

3.2.c. (2 points) **Short answer:** Additive attention defines the attention weights as:

$$\hat{s}_{t,j} = \mathbf{w}_s^T \tanh(\mathbf{W}_s[\mathbf{k}_j; \mathbf{q}_t]), \forall j, t$$

$$\hat{\mathbf{a}}_t = \text{softmax}(\hat{\mathbf{s}}_t), \forall t$$

where the parameters are $\mathbf{w}_s \in \mathbb{R}^{d_k}$ and $\mathbf{W}_s \in \mathbb{R}^{d_k \times 2d_k}$, dimensionality d_k is a hyperparameter, and $[\mathbf{a}; \mathbf{b}]$ is the concatenation of vectors $\mathbf{a}$ and $\mathbf{b}$. Do there exist learnable parameters $\mathbf{w}_s, \mathbf{W}_s$, that can be learned, such that $\hat{s}_{t,j}$ will approximately equal the angle θ between the two vectors $\mathbf{k}_j, \mathbf{q}_t$, or to $\cos(\theta)$? (Briefly justify your answer—a formal proof is not required.)

3.3. Self-attention is typically computed via matrix multiplication. Here we consider multi-headed attention without a causal attention mask.

$$\mathbf{X} = [\mathbf{x}_1, \dots, \mathbf{x}_N]^T$$

$$\mathbf{V}^{(i)} = \mathbf{X} \mathbf{W}_v^{(i)}$$

$$\mathbf{K}^{(i)} = \mathbf{X} \mathbf{W}_k^{(i)}$$

$$\mathbf{Q}^{(i)} = \mathbf{X} \mathbf{W}_q^{(i)}$$

$$\mathbf{S}^{(i)} = \mathbf{Q}^{(i)} (\mathbf{K}^{(i)})^T / \sqrt{d_k}$$

$$\mathbf{A}^{(i)} = \text{softmax}(\mathbf{S}^{(i)})$$

$$\mathbf{X}'^{(i)} = \mathbf{A}^{(i)} \mathbf{V}^{(i)}$$

$$\mathbf{X}' = \text{concat}(\mathbf{X}'^{(1)}, \dots, \mathbf{X}'^{(h)})$$

where N is the sequence length, h is the number of attention heads, and each row involving i is defined $\forall i \in \{1, \dots, h\}$.

3.3.a. (3 points) **Short answer:** Is the score matrix $\mathbf{S}^{(i)}$ always symmetric? If yes, show that it is. If not, describe a condition that would ensure it is symmetric.

- 3.3.b. (4 points) **Short answer:** Suppose we have two attention heads, $h = 2$, we let $d_k = d_m/h$, and we have a single input $\mathbf{X}$. Let $\mathbf{X}'$ be the output of multi-headed attention on $\mathbf{X}$ with the parameters:

$$\mathbf{W}_v^{(1)}, \mathbf{W}_k^{(1)}, \mathbf{W}_q^{(1)}, \mathbf{W}_v^{(2)}, \mathbf{W}_k^{(2)}, \mathbf{W}_q^{(2)} \in \mathbb{R}^{d_m \times d_k}$$

Now suppose we take those same parameters and concatenate along the rows to yield new parameters:

$$\mathbf{W}'_v = \text{concat}(\mathbf{W}_v^{(1)}, \mathbf{W}_v^{(2)}), \mathbf{W}'_k = \text{concat}(\mathbf{W}_k^{(1)}, \mathbf{W}_k^{(2)}), \mathbf{W}'_q = \text{concat}(\mathbf{W}_q^{(1)}, \mathbf{W}_q^{(2)}) \in \mathbb{R}^{d_m \times d_m}$$

And let $\mathbf{X}''$ be the output of single-headed attention on $\mathbf{X}$ with the parameters $\mathbf{W}'_v, \mathbf{W}'_k, \mathbf{W}'_q$.

In this case, does $\mathbf{X}'' = \mathbf{X}'$? Justify your answer.

4 Sliding Window Attention (11 points)

- 4.1. The simplest way to define sliding window attention is by setting the causal mask $\mathbf{M}$ to only include a window of $\frac{1}{2}w + 1$ tokens, where w determines the window size, with the rightmost window element being the current token (i.e. on the diagonal). Then our attention computation is:

$$\mathbf{X}' = \text{softmax}((\mathbf{Q}\mathbf{K}^T / \sqrt{d_k}) + \mathbf{M})\mathbf{V} \quad (1)$$

For example, if we have a sequence of length $N = 6$, and size $w = 4$, then our mask matrix is:

$$\mathbf{M} = \begin{bmatrix} 0 & -\infty & -\infty & -\infty & -\infty & -\infty \\ 0 & 0 & -\infty & -\infty & -\infty & -\infty \\ 0 & 0 & 0 & -\infty & -\infty & -\infty \\ -\infty & 0 & 0 & 0 & -\infty & -\infty \\ -\infty & -\infty & 0 & 0 & 0 & -\infty \\ -\infty & -\infty & -\infty & 0 & 0 & 0 \end{bmatrix}$$

- 4.1.a. (1 point) **Short answer:** If we implement sliding window using the matrix multiplications described in Equation 1, what is the time complexity in terms of N and w ? Let d_k be a constant that does not need to be included in your answer. (For this and subsequent questions, assume that the cost of multiplying two matrices $\mathbf{X} \in \mathbb{R}^{m \times n}$ and $\mathbf{Y} \in \mathbb{R}^{n \times p}$ is $O(mnp)$.)

- 4.1.b. (1 point) **Short answer:** If we implement sliding window using the matrix multiplications described in Equation 1, what is the space complexity in terms of N and w ? Let d_k be a constant that does not need to be included in your answer.

- 4.2. Let's define causal sliding window attention again, but this time, compute $\mathbf{X}'$ from Equation (1) more efficiently.

Pseudocode: Fill in the blanks with pseudocode/math to create a function that takes in the queries, keys, and values and the window size w and computes the $\mathbf{X}'$ of Equation (1):

SLIDINGWINDOWATTENTION($\mathbf{Q}, \mathbf{K}, \mathbf{V}, w$)

Your pseudocode/math must have lower asymptotic computational than the naive matrix multiplication approach described above. The function `softmax(x)` can be applied to a vector $\mathbf{x}$ or a matrix $\mathbf{X}$ row-wise. The function `tensor()` can be used to construct vectors, matrices, tensors of arbitrary shape specified by a `shape` parameter and initializes their values to a scalar `init_values`. In the pseudocode below, X_i means indexing into the vector / matrix X at index i . The `range()` function follows standard Python convention. For instance, `range(0, N)` iterates over the values 0 (inclusive) to N (exclusive).

```
def SlidingWindowAttention(Q, K, V, w):
    N, d_k = Q.shape()
    X' = tensor(shape=(N, d_k), init_values=0)
    w' = w//2 + 1

    for j in range(0, N):

        # a stores attention scores only for this local window
        a = tensor(shape=__ (1a) __, init_values=__ (1b) __)

        # loop over local window indices
        for i in range(__ (2) __):
            # map local index to the corresponding token index
            token_idx = __ (3) __
            # ignore if token_idx is out of bounds
            if __ (4) __ :
                # compute attention score
                a_i = __ (5) __

        a = softmax(a)

        # loop again to build the weighted sum
        for i in range(__ (2) __):
            token_idx = __ (3) __
            if __ (4) __ :
                # accumulate weighted value
                X'_j += __ (6) __

        # delete a and trigger garbage collection
        # so it can be reused in the next iteration
        del a
        gc.collect()
    return X'
```

4.2.a. (1 point) Write pseudocode/math to fill in blank **(1a)**.

4.2.b. (1 point) Write pseudocode/math to fill in blank **(1b)**.

4.2.c. (1 point) Write pseudocode/math to fill in blank **(2)**. Note that both blanks will be filled with the same line.

4.2.d. (1 point) Write pseudocode/math to fill in blank **(3)**. Note that both blanks will be filled with the same line.

4.2.e. (1 point) Write pseudocode/math to fill in blank **(4)**.

4.2.f. (1 point) Write pseudocode/math to fill in blank **(5)**.

4.2.g. (1 point) Write pseudocode/math to fill in blank **(6)**.

- 4.2.h. (1 point) **Short answer:** What is the space complexity of your pseudocode in terms of N and w ? Note both N and w need to be in your answer.



- 4.2.i. (1 point) **Short answer:** What is the time complexity of your pseudocode in terms of N and w ? Note both N and w need to be in your answer.



5 Programming: RoPE and GQA (22 points)

Introduction

In this section, you will take a run-of-the-mill GPT model and upgrade it to incorporate two of the key ingredients found in state-of-the-art large language models (LLMs), such as [LLAMA-2](#).

The first ingredient are rotary position embeddings (RoPE). These will replace the existing absolute position embeddings with a relative position embedding that rotates small segments of each key and query vector.

The second ingredient is grouped-query attention (GQA). Although the GQA mechanism is fundamentally still causal attention, it enables the model to use less memory and run faster.

You will experiment with how these two model improvements lead to changes in model performance. And you will even evaluate how they perform in tandem.

Upon completion of this section, you will unfortunately not be able to claim to have trained a *large* language model, for the dataset we provide here (the complete works of Shakespeare) is rather small if not trite. However, you can reasonably claim to have built your own LLAMA-2 model.

Dataset

The dataset for this homework is a collection of the complete works of Shakespeare. The dataset file is `input.txt`, and is around 1.1MB in size.

Starter Code

The starter code was originally authored by [Andrej Karpathy](#), of OpenAI fame, and released as [minGPT](#). It offers a clear glimpse into the inner workings of a GPT model. We have simplified the codebase and provided to you a modified version. Ours contains the following files:

```
hw1/  
  requirements.txt  
  input.txt  
  chargpt.py  
  mingpt/  
    model.py  
    trainer.py  
    utils.py  
  test_model.py
```

Here is what you will find in each file:

1. `requirements.txt`: A list of packages that need to be installed for this homework. This homework only requires 2 packages - `torch` and `einops`.
2. `input.txt`: The dataset—the works of Shakespeare.
3. `chargpt.py`: The main entry point used to train your transformer. It can be run with the command `python chargpt.py`. Append flags to this command to adjust the transformer configuration. You will modify this file when changing prompts.

4. `mingpt/model.py`: Is the other file you will modify (more extensively) for this homework. This file contains the construction of the GPT model. A vanilla, working transformer implementation is already provided. You will implement the classes `RotaryPositionalEmbeddings` and `GroupedQueryAttention`. You will also need to make changes to the class `CausalSelfAttention` while implementing RoPE. (Hint: Locations in the code where changes ought to be made are marked with a **TODO**.)
5. `mingpt/trainer.py`: Code for the training loop of the transformer.
6. `mingpt/utils.py`: Helper functions for saving logs and configs.
7. `test_model.py`: A file containing unit tests (the same ones used on Gradescope). To run them, simply execute `python test_model.py` in your terminal. Please note however that from an assessment perspective, we will continue to manually grade all of your code submissions and that manual evaluation will be the bulk of your grade on the programming portion of the homework.

Flags

All the parameters printed in the config can be modified by passing flags to `chargpt.py`. Table 1 contains a list of flags you may find useful while implementing HW1. You can change other parameters as well in a similar manner. Simply specify the config node (i.e. one of `{system,data,model,trainer}`), followed by a period `.`, followed by the parameter you wish to modify.

Configuration Parameter	Example Flag Usage
Model sequence length	<code>--data.block_size=16</code> (<code>model.block_size</code> is autoset based on this flag)
Directory where model is stored	<code>--system.work_dir=out/new_chargpt</code>
Number of query heads (hyperparameter for GQA)	<code>--model.n_query_head=6</code>
Number of key-value heads (hyperparameter for GQA)	<code>--model.n_kv_head=3</code> (<code>n_query_head</code> must be divisible by <code>n_kv_head</code>) (For standard multi-head attention <code>n_query_head = n_kv_head</code>)
Directory from which to load a model trained in a previous run	<code>--model.pretrained_folder=out/chargpt3</code>
Whether to enable RoPE embeddings	<code>--model.rope=True</code>
Number of iterations to train the model	<code>--trainer.max_iters=200</code>
Device type (useful for debugging), one of "cpu", "cuda"	<code>--trainer.device=cpu</code>

Table 1: Useful flags for `chargpt.py`

Model

The default model in `chargpt.py` is a GPT model with 6 transformer layers. Each attention layer uses $h = 6$ attention heads. The maximum sequence length is $N = 16$. Because the vocabulary is comprised of only characters, the vocabulary size is only 65. The embedding dimension is $d_{model} = 192$ and the key/value/query dimension size is $d_k = d_{model}/h = 32$.

Various Files Included to Disable AI Coding Tools

In the handout, we've included various files such as `.cursor` and `.vscode` to disable AI coding tools (e.g. Cursor and Copilot, respectively). Since we know that you may already have these coding agents for other projects, these files are present in order to make it easier for you to do your homework without AI assistance for the Slot A deadline. As a reminder, submitting AI-assisted work to Slot A, or claiming AI-assisted work is human-only work, will be treated as an Academic Integrity Violation and may result in severe penalties, including failure in the course.

Google Colab

Colab provides a free T4 GPU for code execution, albeit with a time limitation that may result in slower training. In the event of GPU depletion on Colab, options include waiting for GPU recovery, switching Google accounts, purchasing additional GPU resources (\$10 for Colab Premium), switching to Kaggle, or switching to a cloud provider (such as GCP or AWS).

Kaggle

Kaggle provides 30 hours of free T4 or V100 GPU runtime per week, which is sufficient for completing this homework. If you wish to use Kaggle, follow the steps below to set up a Kaggle environment.

1. Enable GPU Access

- After creating an account, go to <https://www.kaggle.com/settings> and complete the steps for **phone verification**
- Go to <https://www.kaggle.com/code> and click on **New Notebook**
- In the right-hand sidebar, under the **Session options** tab:
 - **Tip:** Switch the Accelerator to GPU only when you're ready to run your code, to optimize resource usage
 - Toggle **Accelerator** to ****GPU (T4 or V100)****
 - Set **Persistence** to ****Variables and Files****
 - Make sure to switch **Internet** to ****Internet on****

2. Upload Homework Files (Optional if following notebook setup)

- Navigate to the **File** drop down in the upper left corner to import the notebook for Kaggle.
- On the right side under **Input**:
 - Click on **Upload** and **New Dataset**
 - Zip all of your starter code and data files together and upload them here
- Once uploaded, the files will be available under **Input** where you can directly copy the paths to fill in the notebook

Local

We recommend debugging your code locally with `test_model.py`, and with your own test cases. We do not recommend training your code on CPU.

Rotary Position Embeddings (RoPE)

In this section, you will implement Rotary Position Embeddings (RoPE) (Su et al., 2021).

Background: Absolute position embeddings are added to the word embeddings in the first layer of a standard Transformer language model. Subsequent layers propagate position information up from the bottom.

Traditional attention is defined as below.

$$\begin{aligned} \mathbf{q}_j &= \mathbf{W}_q^T \mathbf{x}_j, \forall j \\ \mathbf{k}_j &= \mathbf{W}_k^T \mathbf{x}_j, \forall j \\ s_{t,j} &= \mathbf{k}_j^T \mathbf{q}_t / \sqrt{d_k}, \forall j, t \\ \mathbf{a}_t &= \text{softmax}(\mathbf{s}_t), \forall t \end{aligned}$$

where $d_k = |\mathbf{k}_j|$ is the size of the query/key/value vectors.

RoPE: Rotary Position Embeddings (RoPE) (Su et al., 2021) incorporate positional information directly into the attention computation, in every layer. If the input to the next attention layer is $\mathbf{X} = [\mathbf{x}_1, \dots, \mathbf{x}_N]^T$, then we introduce two functions $f_q(\mathbf{x}_j, j)$ and $f_k(\mathbf{x}_j, j)$, which compute the position-aware queries and keys respectively. Then the attention scores are computed as below:

$$\begin{aligned} \mathbf{q}_j &= \mathbf{W}_q^T \mathbf{x}_j, \forall j & \mathbf{k}_j &= \mathbf{W}_k^T \mathbf{x}_j, \forall j \\ \tilde{\mathbf{q}}_j &= \mathbf{R}_{\Theta,j} \mathbf{q}_j & \tilde{\mathbf{k}}_j &= \mathbf{R}_{\Theta,j} \mathbf{k}_j \\ s_{t,j} &= \tilde{\mathbf{k}}_j^T \tilde{\mathbf{q}}_t / \sqrt{d_k}, \forall j, t \\ \mathbf{a}_t &= \text{softmax}(\mathbf{s}_t), \forall t \end{aligned}$$

where $\mathbf{W}_k, \mathbf{W}_q \in \mathbb{R}^{d_{\text{model}} \times d_k}$. Herein we use $d = d_k$ for brevity.

To implement this efficiently in PyTorch, we want to construct a new matrix $\tilde{\mathbf{Y}} = g(\mathbf{Y}; \Theta)$ such that $\tilde{\mathbf{Y}}_{m,\cdot} = \mathbf{R}_{\Theta,m} \mathbf{y}_m$ for a matrix of embeddings $\mathbf{Y} = [\mathbf{y}_1, \dots, \mathbf{y}_N]^T \in \mathbb{R}^{N \times d}$, and $\Theta = \{\theta_i = 10000^{-2(i-1)/d}, i \in [1, 2, \dots, d/2]\}$ (in practice this $\mathbf{Y}$ would be either the queries $\mathbf{Q}$ or the keys $\mathbf{K}$). We can construct the new matrix as follows:

$$\begin{aligned} \tilde{\mathbf{Y}} &= g(\mathbf{Y}; \Theta) \\ &= \left[\begin{array}{ccc|ccc} Y_{1,1} & \cdots & Y_{1,\frac{d}{2}} & Y_{1,\frac{d}{2}+1} & \cdots & Y_{1,d} \\ \vdots & & \vdots & \vdots & & \vdots \\ Y_{N,1} & \cdots & Y_{N,\frac{d}{2}} & Y_{N,\frac{d}{2}+1} & \cdots & Y_{N,d} \end{array} \right] \odot \left[\begin{array}{ccc|ccc} \cos 1\theta_1 & \cdots & \cos 1\theta_{\frac{d}{2}} & \cos 1\theta_1 & \cdots & \cos 1\theta_{\frac{d}{2}} \\ \vdots & & \vdots & \vdots & & \vdots \\ \cos N\theta_1 & \cdots & \cos N\theta_{\frac{d}{2}} & \cos N\theta_1 & \cdots & \cos N\theta_{\frac{d}{2}} \end{array} \right] \\ &+ \left[\begin{array}{ccc|ccc} -Y_{1,\frac{d}{2}+1} & \cdots & -Y_{1,d} & Y_{1,1} & \cdots & Y_{1,\frac{d}{2}} \\ \vdots & & \vdots & \vdots & & \vdots \\ -Y_{N,\frac{d}{2}+1} & \cdots & -Y_{N,d} & Y_{N,1} & \cdots & Y_{N,\frac{d}{2}} \end{array} \right] \odot \left[\begin{array}{ccc|ccc} \sin 1\theta_1 & \cdots & \sin 1\theta_{\frac{d}{2}} & \sin 1\theta_1 & \cdots & \sin 1\theta_{\frac{d}{2}} \\ \vdots & & \vdots & \vdots & & \vdots \\ \sin N\theta_1 & \cdots & \sin N\theta_{\frac{d}{2}} & \sin N\theta_1 & \cdots & \sin N\theta_{\frac{d}{2}} \end{array} \right] \end{aligned}$$

Or more compactly:

$$\mathbf{C} = \left[\begin{array}{ccc|ccc} 1\theta_1 & \cdots & 1\theta_{\frac{d}{2}} & 1\theta_1 & \cdots & 1\theta_{\frac{d}{2}} \\ \vdots & & \vdots & \vdots & & \vdots \\ N\theta_1 & \cdots & N\theta_{\frac{d}{2}} & N\theta_1 & \cdots & N\theta_{\frac{d}{2}} \end{array} \right]$$

$$\tilde{\mathbf{Y}} = g(\mathbf{Y}; \Theta)$$

$$= \left[\mathbf{Y}_{:,1:d/2} \mid \mathbf{Y}_{:,d/2+1:d} \right] \odot \cos(\mathbf{C})$$

$$+ \left[-\mathbf{Y}_{:,d/2+1:d} \mid \mathbf{Y}_{:,1:d/2} \right] \odot \sin(\mathbf{C})$$

Now we can compute RoPE embeddings efficiently as below:

$$\begin{aligned} \mathbf{Q} &= \mathbf{XW}_q & \mathbf{K} &= \mathbf{XW}_k \\ \tilde{\mathbf{Q}} &= g(\mathbf{Q}; \Theta) & \tilde{\mathbf{K}} &= g(\mathbf{K}; \Theta) \\ \mathbf{S} &= \tilde{\mathbf{Q}}\tilde{\mathbf{K}}^T / \sqrt{d_k} \\ \mathbf{A} &= \text{softmax}(\mathbf{S}) \end{aligned}$$

You do not have to understand all the math in the paper, but you may go through it to understand the intuition behind RoPE.

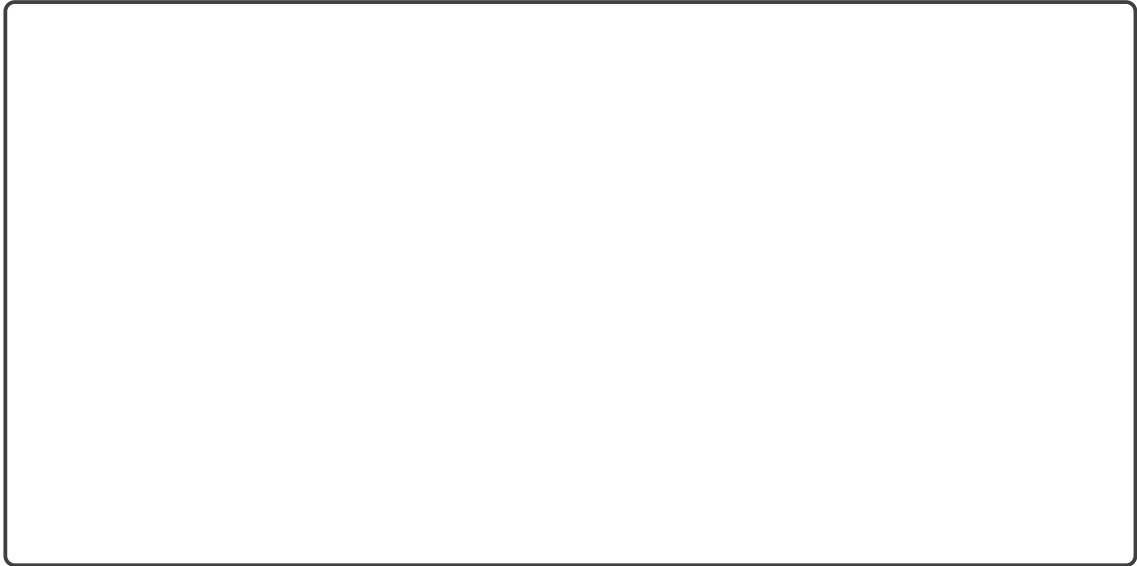
Implementation: You will implement RoPE within `minGPT`. To do so, you should make changes to the `RotaryPositionalEmbeddings` and the `CausalSelfAttention` classes in `minGPT/model.py`. Within `RotaryPositionalEmbeddings`, you will first implement `_build_cache`, and within the forward computation, your first steps should be building the cache if it has not been built yet.

RoPE Empirical Questions

For all empirical plots in this section, you are free to use wandb, matplotlib, or another similar plotting tool.

- 5.1. (2 points) Provide a sample of text generation from your RoPE model after **600 iterations** of training with a **sequence length of 16**. Condition the generated sample on the first line of your favorite Shakespeare play. Do not use the default line provided and do not modify the provided `max_new_tokens` value in the function call.

[Expected runtime on Colab T4: approximately 3 minutes]



- 5.2. (2 points) Provide a sample of text generation from your RoPE model after **1200 iterations**, consisting of 600 iterations with a **sequence length of 16**, followed by an additional 600 iterations with a **sequence length of 256** on the same model. Condition the generated sample on the first line of your favorite Shakespeare play. Do not use the default line provided.

[Expected runtime on Colab T4: approximately 5 minutes]



- 5.3. (2 points) Inspect the code in `model.py` that performs the generation, `GPT.generate()`. Does this include a KV-cache? If yes, identify which lines of code are doing so, and how they work. If not, explain how the model handles attention at each timestep.

- 5.4. (4 points) Plot the training loss for both your RoPE implementation and the vanilla minGPT over **1200 total training iterations** on the **same plot**, and train as follows: 600 iterations with a **sequence length of 16**, and then continue training this model for another 600 iterations, but now with a **sequence length of 256**.

[Expected runtime on Colab T4 to run both: approximately 10 minutes]

Grouped Query Attention (GQA)

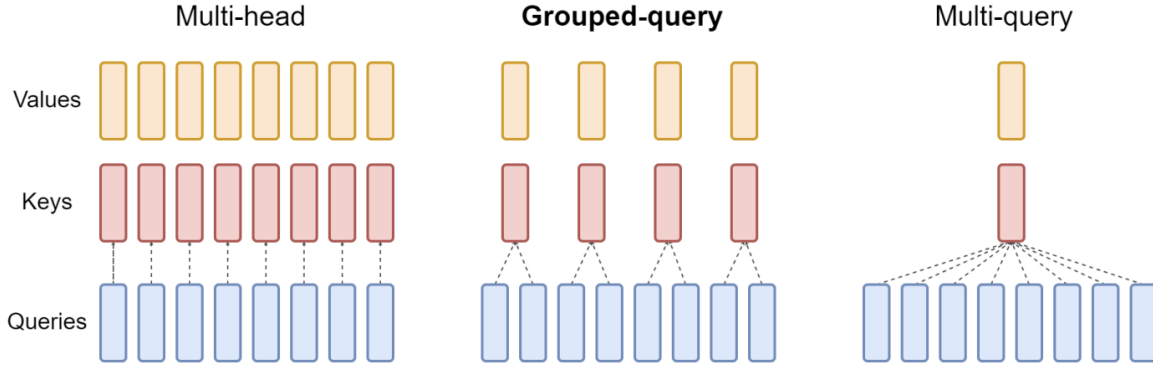


Figure 1: Schematic representation of attention mechanisms, showcasing Multi-head attention with individual keys and values for each head, Grouped-query attention with queries grouped to share common keys and values, and Multi-query attention utilizing a singular key and value for all queries.

In this section, you will implement Grouped Query Attention (GQA) ([Ainslie et al., 2023](#)).

GQA: Grouped Query Attention (GQA) is a technique in neural network architectures that modifies the attention mechanism used in models such as transformers. It involves dividing the query heads into groups, each sharing a single key head and value head. This approach can interpolate between Multi-Query Attention (MQA) and Multi-Head Attention (MHA), offering a balance between computational efficiency and model quality [Figure 1].

We define the following variables:

- h_q : Number of query heads.
- h_{kv} : Number of key/value heads.
- $g = h_q/h_{kv}$: the size of each group (i.e. number of query vectors per key/value vector).
Note that we assume h_q is divisible by h_{kv} .

Our parameter matrices for GQA are all the same size: $\mathbf{W}_q^{(i,j)}, \mathbf{W}_k^{(i)}, \mathbf{W}_v^{(i)} \in \mathbb{R}^{d_{model} \times d_k}$ where $i \in \{1, \dots, h_{kv}\}$, $j \in \{1, \dots, g\}$, and $d_k = d_{model}/h_q$. However, we now have different numbers of query, key, and value heads:

$$\begin{aligned} \mathbf{X} &= [\mathbf{x}_1, \dots, \mathbf{x}_T]^T \\ \mathbf{V}^{(i)} &= \mathbf{XW}_v^{(i)}, \forall i \in \{1, \dots, h_{kv}\} \\ \mathbf{K}^{(i)} &= \mathbf{XW}_k^{(i)}, \forall i \in \{1, \dots, h_{kv}\} \\ \mathbf{Q}^{(i,j)} &= \mathbf{XW}_q^{(i,j)}, \forall i \in \{1, \dots, h_{kv}\}, \forall j \in \{1, \dots, g\} \end{aligned}$$

Above, we define g times more query vectors than key/value vectors. Then we compute the scaled dot-product between each query vector (i, j) and its corresponding key (i) to get a similarity score. The similarity scores are used to compute an attention matrix, but with only h_{kv} heads

$$\begin{aligned}
\mathbf{S}^{(i,j)} &= \mathbf{Q}^{(i,j)} (\mathbf{K}^{(i)})^T / \sqrt{d_k}, \quad \forall i \in \{1, \dots, h_{kv}\}, \forall j \in \{1, \dots, g\} \\
\mathbf{A}^{(i,j)} &= \text{softmax}(\mathbf{S}^{(i,j)}), \quad \forall i \in \{1, \dots, h_{kv}\}, \forall j \in \{1, \dots, g\} \\
\mathbf{X}'^{(i,j)} &= \mathbf{A}^{(i,j)} \mathbf{V}^{(i)}, \quad \forall i \in \{1, \dots, h_{kv}\}, \forall j \in \{1, \dots, g\} \\
\mathbf{X}' &= \text{concat}(\mathbf{X}'^{(i,j)}), \quad \forall i \in \{1, \dots, h_{kv}\}, \forall j \in \{1, \dots, g\} \\
\mathbf{X} &= \mathbf{X}' \mathbf{W}_o \quad (\text{where } \mathbf{W}_o \in \mathbb{R}^{d_{\text{model}} \times d_{\text{model}}})
\end{aligned}$$

Implementation Details: You will implement GQA in the `GroupedQueryAttention` class in `mingpt/model.py`. Much of your code will be similar to that in `CausalSelfAttention`.

Hint: You may find it easier to implement `GroupedQueryAttention` in a similar way to `CausalSelfAttention`.

- **Initialization:**
 - Familiarize yourself with the configuration settings that initialize the attention mechanism, including the number of query heads, key/value heads, and embedding dimensions. Note that `config` is defined in the `get_config` method in `chargpt.py`.
 - Ensure the embedding dimension is divisible by the number of query and key/value heads.
- **Regularization:**
 - Incorporate dropout layers for attention and residuals to prevent overfitting.
- **Dimensionality and Projections:**
 - Implement the linear projection layers for queries, keys, and values, considering the dimensionality constraints of the parameter matrices *and* the grouped nature of the mechanism.
- **Rotary Positional Embeddings:**
 - If rotary positional embeddings are enabled, integrate RoPE with query and key projections.
- **Forward Pass:**
 - In the forward method, transform the input according to the query, key, and value projections.
 - Apply the attention mechanism by computing grouped scaled dot-product attention
 - Mask the attention to ensure causality (preventing future tokens from being attended to).
 - Aggregate the attention with the values and project the output back to the embedding dimension.
- **Memory Efficiency:**
 - Monitor and record the CUDA memory allocation before and after the attention operation to analyze the memory efficiency of the GQA. A reference code to monitor memory is present in `CausalSelfAttention` class.

GQA Empirical Questions

The questions below assume you are using absolute position embeddings, not RoPE.

- 5.5. (4 points) Plot the **average time taken** to compute attention per iteration in milliseconds across $\{1, 2, 3, 6\}$ number of key heads with a **sequence length of 16** over **200 iterations**.

[Expected runtime on Colab T4: approximately 1 minute]



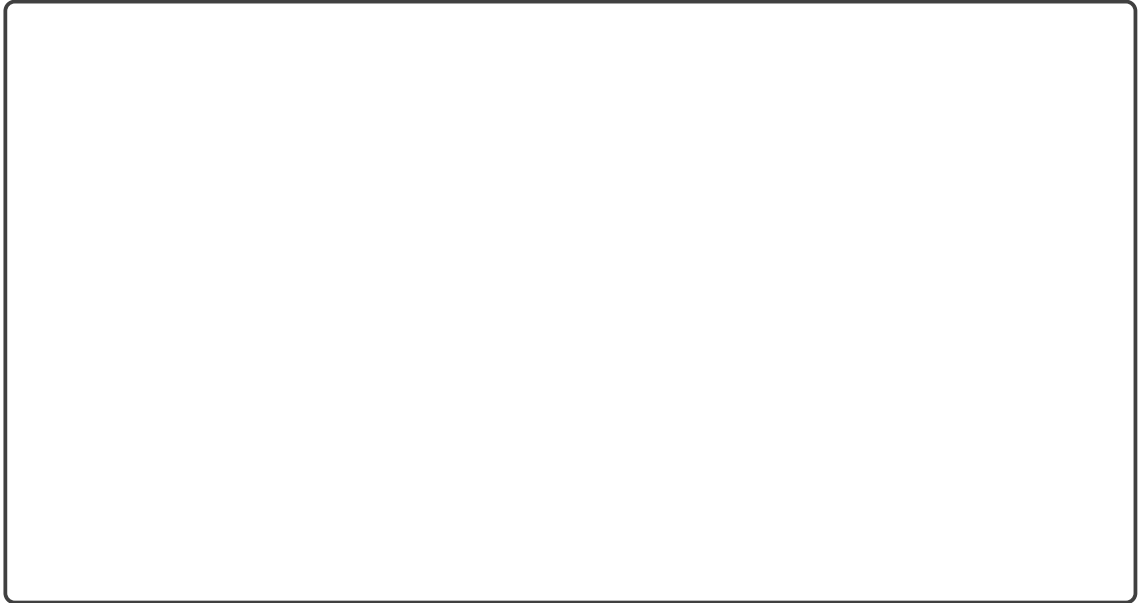
- 5.6. (4 points) Plot both the training loss of your GQA implementation with 2 key heads and the original (multi head attention) minGPT over **200 iterations** with a **sequence length of 16** on the **same plot**.

[Expected runtime on Colab T4 to run both: approximately 6 minutes]



- 5.7. (4 points) Plot the following four configurations on the **same plot**: 1. vanilla minGPT (no RoPE nor GQA), 2. RoPE only (no GQA), 3. GQA only (no RoPE), 4. RoPE and GQA. For each, plot the training loss over **1200 total training iterations**: 600 iterations with a **sequence length of 16**, followed by 600 iterations with a **sequence length of 256**.

[Expected runtime on Colab T4 to run all four: approximately 16 minutes]



6 Code Upload (1 points)

- 6.1. (1 point) Which files did you upload to the appropriate programming slot on Gradescope? *Hint:* For this homework, you should upload only `model.py`.

A large, empty rectangular box with a thin black border, intended for the user to provide the file names they uploaded for this question.

7 Collaboration Questions (2 points)

After you have completed all other components of this assignment, report your answers to these questions regarding the collaboration policy. Details of the policy can be found in the syllabus.

- 7.1. (1 point) Did you collaborate with anyone on this assignment? If so, list their name or Andrew ID and which problems you worked together on.

- 7.2. (1 point) Did you find or come across code that implements any part of this assignment? If so, include full details.