Natural Language Processing Assignment 2

April 1, 2025

Due date: May 6 11:59 PM. These questions require thoughtful responses, but concise answers are expected. While group discussions are encouraged, the final work must be completed independently. Please submit a package consisting of a report (in LaTeX or Markdown) and source code with necessary annotations. Your report should include your implementation ideas, observations, and must not exceed 8 pages. Ensure the report contains your name and student number. If you have any questions regarding this homework, feel free to contact the TAs or discuss them during office hours or in the WeChat group NLP@THU2025.

1 Prompt Engineering (5 Points)

1.1 Designing a Unique and Challenging Prompt (2.5 Points)

Design a prompt that explicitly tests two specific LLMs, ensuring that only one can generate the correct answer while the other fails. The problem must have a single objective and easily verifiable answer and not be solvable by the other model. Test the prompt using two of the following models with their respective requirements:

- Deepseek-V3 (with deep thinking)
- GPT-4o, o1 or o3
- Gemini 2.5 Pro Experimental 03-25
- Claude-3.7-sonnet

Requirements:

1. Design a question. Ensure the question has a single objective and easily verifiable answer. If necessary, provide the source/explanation for the answer.
2. Repeat the prompt three times to check for consistency in the result. Provide a total of six screenshots in your report:
    - Three screenshots from one model showing correct answers.
    - Three screenshots from another model showing incorrect answers.
3. The prompt and correct answer must be in English.

1.2 In-Context Learning (2.5 Points)

In-context learning means the model can learn from the information provided in the context. In this question, you need to design a prompt to solve reasoning questions.

Requirements

1. Randomly select three reasoning-based questions from the GSM-8K dataset.
2. Design a prompt to solve the three chosen questions by In-Context Learning. (Solve three questions within one prompt)

3. Ensure the output is in a structured format, such as JSON or YAML, and includes the following fields:
    - **Question ID**
    - **Reasoning Process** (a list)
    - **Final Answer**
    - **Difficulty Classification**
4. Provide two prompts in your report, both producing outputs in structural format as mentioned above.
    - **Prompt 1**: Uses in-context learning with several in-context demonstrations (few-shot learning).
    - **Prompt 2**: Does not use in-context learning (zero-shot learning).
5. Compare the results of the two prompts and briefly explain the pros and cons of each prompt.

2 NLP Architecture Analysis (8 points + 2 bonus points)

In class, we introduced the Transformer Architecture [1] and its extension, the GPT Architecture, which were designed to overcome the bottlenecks of recurrent neural networks (RNNs), particularly their inability to handle long-range dependencies efficiently. One of the main limitations of RNNs is their sequential nature, which makes it difficult to model long-term dependencies and parallelize computations. Transformers address this issue using the self-attention mechanism, which allows global context modeling by attending to all tokens in the input sequence simultaneously. This innovation, combined with mechanisms like positional encoding and multi-head attention, has revolutionized NLP. Beyond the original Transformer and GPT architectures, several other models and techniques have been proposed to further optimize performance, memory usage, and computational efficiency. These include:

- **GQA (Grouped Query Attention)** [2]: A variation of multi-head attention that reduces memory cost by sharing key and value vectors between different attention heads.
- **Mamba-1 [3] and Mamba-2** [4]: Mamba models improve the efficiency of language models by replacing the expensive attention mechanism with selective state-space models (SSMs). Mamba-2 extends Mamba-1 by increasing the state size and improving the implementation's efficiency.
- **GLA (Gated Linear Attention)** [5]: An efficient attention mechanism that replaces softmax-based attention with linear attention, incorporating gating mechanisms to control information flow and improving efficiency by reformulating attention into a recurrent architecture.

In this question, you are required to calculate (1) the model size, (2) the KV Cache size and (3) the FLOPs in a forward pass of a language model. To help you better understand the underlying concepts, here is more context for each key metric:

- **Model Size**:
    - **Units**: Parameters (often expressed in millions or billions, e.g., M or B)
    - **Description**: The total number of trainable parameters in a neural network, which determines memory storage requirements and affects computational efficiency. It includes both embedding parameters and transformer parameters.
        * **Embedding Parameters**: These are the parameters used in the embedding layers of the model, which convert discrete input elements (e.g., words, tokens) into dense vector representations.

    - **Transformer Parameters**: These are the parameters used within the transformer architecture, including those in the attention mechanisms, feed-forward networks, and layer normalization components.

- **FLOPs (Floating Point Operations)**:
  - **Units**: FLOPs (often expressed as billions or trillions of FLOPs, e.g., GFLOPs or TFLOPs)
  - **Description**: FLOPs quantify the number of floating-point operations (additions or multiplications) performed in a computation. By counting the number of FLOPs involved in each forward/backward pass of a neural network, we can measure the network's computational cost.
- **KV Cache Size**:
  - **Units**: Bytes (or MB/GB depending on scale)
  - **Description**: During autoregressive generation, the keys and values in the attention layer of past tokens are cached to avoid recomputing them at each time step. This storage, known as the KV cache, can lead to considerable memory and I/O costs.

2.1 Assumptions

For simplification, we make the following assumptions:

- Model parameters and KVs are represented in `bfloat16` precision.
- The number of FLOPs for matrix multiplication between two tensors with shapes (a, b) and (b, c) is approximated as $2abc$.
- The model is a standard GPT-2 model[1], which means the model:
  - does not use SwiGLU FFN [6], which is common practice nowadays (see Qwen2.5's code for implementation). GPT-2's FFN consists of two linear layers instead.
  - uses an attention head size that equals d/n_h.
  - uses tied embeddings, which means the LM head (a.k.a. output embedding layers) shares the same parameters with the input embedding layer (a.k.a. embedding table).
  - uses learned position embeddings.
- The FLOPs of the following operations are considered negligible and can be ignored:
  - Normalization layers
  - Addition with residual connections
  - The softmax function
  - The embedding lookup operation
  - The construction of the causal attention mask.
- You are free to make additional reasonable assumptions as long as they do not significantly affect the total number of FLOPs or memory usage.

2.2 Notations

You must use the notations given in Tables 1, 2, and 3 in your report, and your final answer must be expressed with those variables.

[1] see `https://github.com/karpathy/nanoGPT` for a code implementation.

Variable	Description
L	Input sequence length
V	Vocabulary size
d	Hidden size
N_{layer}	Number of layers
n_q	Number of query heads
n_{kv}	Number of key-value heads, i.e., the number of groups in GQA.
d_{ff}	Intermediate dimension of the feed-forward network (FFN)

Table 1: List of variables used in GPT.

Variable	Description
L	Input sequence length
V	Vocabulary size
d	Hidden size
N_{layer}	Number of layers
E	Expansion ratio, defaults to 2.
P	Dimensionality of the x, defaults to 64.
N	State size, defaults to 128

Table 2: List of variables used in Mamba-2.

Variable	Description
L	Input sequence length
V	Vocabulary size
d	Hidden size
N_{layer}	Number of layers
n_h	Number of attention heads
d_v	Dimension of value vectors
d_k	Dimension of key vectors
d_{ff}	Intermediate dimension of the feed-forward network (FFN)

Table 3: List of variables used in GLA.

2.3 Requirements

1. Compute the model size, KV cache size and FLOPs of the following models in a forward pass. Show the derivation of your calculations for each metric (Model size, KV cache size and FLOPs). Summarize your results in a table comparing the models side-by-side.

   - GPT (with multi-head attention):
   - GPT with GQA (Grouped Query Attention)
   - Bonus (1 point): Mamba-2
   - Bonus (1 point): GLA (Gated Linear Attention)

   Using the notations above, you must express the model size, KV cache size, and FLOPs, **each with a single expression**.

2. Compare the computational and memory efficiency of the models. Provide a brief analysis of the trade-offs between these architectures.

2.4 Hints

- You can print the number of parameters and FLOPs of an open-source model (e.g., Qwen2.5, GLA, Mamba-2) to verify your answer.

- PyTorch has a built-in function for counting the number of FLOPs in a forward pass.
- Make sure parameters shared by multiple modules are only counted once.
- Mamba-2 does not have FFN layers.
- Make sure you take into account the fact that GPT uses causal masking in the self-attention layer; hence, only the lower half of the attention score matrix is being computed.

3 Model Implementation (7 points)

In this task, you should complete only one of the two subtasks below (3.1 or 3.2) .

3.1 Modifying Qwen2.5

Qwen2.5 is the latest series of Qwen large language models. In research, we often need to modify model architectures to fit specific requirements. In this task, you are asked to make the following modifications to the Qwen2.5 model code.

3.1.1 Learned Position Encoding

Task Replace the rotary positional encoding in Qwen2.5 with learned position encoding, which is added to the hidden representations right after the input embedding layer.

Background Learned position encoding can be formulated as follows. Let the input sequence be $(x_1, \ldots, x_L)$ where x_i is a d-dimensional vector, then, after the input embedding layer and before any transformer layer, we execute:

$$x_i \leftarrow x_i + p_i \quad i = 1, \cdots, L$$

where p_i is the learned position embedding for position i.

3.1.2 DyT (Dynamic Tanh)

Task Replace RMSNorm with DyT.

Background [7] propose to replace all normalization layers with a Dynamic Tanh (DyT) module, which can be more efficient while maintaining performance. Refer to the paper "Transformers without Normalization" for more details.

3.1.3 ReLU-Attention: [8]

Task Replace the softmax operation in attention with a ReLU-Attention.

Background Let a_{ij} denote the attention score from the i-th token to the j-th token. The standard attention is computed as:

$$a_{ij} = \text{softmax}_j\left(\frac{q_i k_j^\top}{\sqrt{d_h}}\right) = \frac{\exp(\tilde{a}_{ij})}{\sum_j \exp(\tilde{a}_{ij})}$$

where $\tilde{a}_{ij} = q_i k_j^\top / \sqrt{d_h}$, $q_i k_j^\top$ is the dot-product between q_i and k_j. ReLU-Attention replaces the softmax function with:

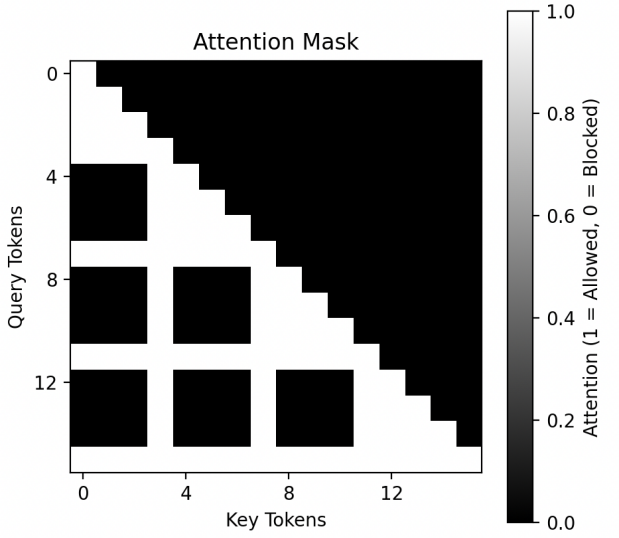
$$a_{ij} = \frac{\text{ReLU}(q_i k_j^\top)}{L}$$

where the division by the sequence length L ensures training stability. See ReLU Attention for more details.

3.1.4 Layer-Dependent Attention Mask

Task Change the standard causal masking to the following attention mask.
Adjust the attention masking strategy as follows:

- The **first layer** applies a **causal attention mask**, restricting each token to attend only to past tokens.
- The **last layer** uses **dilated sliding window attention** with a **window size of 128** and a **dilation factor of 2**, meaning each token attends to a sparse subset of past tokens. Specifically, the token at position t attends to tokens at positions $t, t-2, t-4, t-6, \ldots, t-126$.
- **Middle layers** apply a **custom attention mask**. For example, for a sequence of 16 tokens divided into 4 blocks, the attention mask is shown in Figure 3.1.4. You are required to implement the attention mask for a sequence of L tokens divided into 4 blocks, following the pattern shown in the example.



Requirements:

1. Clone the Qwen2.5 code from HuggingFace transformers[2].
2. Implement the modifications listed above.
3. Your implementation should all be contained in one single file, called `modeling_qwen2.py`. Make sure your model can be a drop-in replacement of the class `Qwen2ForCausalLM` in the official Qwen2.5 implementation in HuggingFace transformers.

Ensure your code is well-documented, with comments explaining each step of your implementation. Even if you are unable to fully complete the task, partially correct or incomplete implementations will still be graded, so make sure to document your approach and reasoning as clearly as possible.

[2] `https://github.com/huggingface/transformers/blob/main/src/transformers/models/qwen2/modeling_qwen2.py`

3.2 Manual Forward of ReGLA

Gated Linear Attention (GLA), introduced by [5], enhances linear attention mechanisms by incorporating data-dependent gating mechanisms inspired by RNNs. However, GLA inherits the saturation problem of sigmoid gates ($g = \sigma(Wx)$), wherein gradients vanish as $\nabla g = g \odot (1 - g)$, particularly when the activation approaches boundary values (0 or 1). This issue hampers effective learning during training.

ReGLA [9] extends GLA by introducing a refining gate mechanism to address the saturation problem and improve gradient flow. This modification interpolates between the lower and upper bounds of gate activations, leading to a more stable training process and enhanced model performance.

Requirements:

1. **Mathematical Formulation**:
    - Read the papers on GLA and ReGLA. Write down the mathematical formulation of the forward pass for both the GLA and ReGLA layers, ensuring all variables involved are clearly defined.[3]
2. **Implementation using Numpy**:
    - Implement the forward function for the ReGLA layer using only the `numpy` library. Your implementation does not need to be efficient.
    - **Parameter Initialization**: Initialize all parameters involved in both GLA and ReGLA layers to the constant value of 0.01 for simplicity during initial testing. Note: In practice, parameters should be initialized using random methods.
    - **Restriction**: You are not allowed to use advanced machine learning libraries such as PyTorch, TensorFlow, or MindSpore.

Your implementation should all be contained in one single file. Ensure your code is well-documented, with comments explaining each step of your implementation. Even if you are unable to fully complete the task, partially correct or incomplete implementations will still be graded, so make sure to document your approach and reasoning as clearly as possible.

4 Scoring Breakdown

- Prompt Engineering: 5 points
- NLP Architecture Analysis: 8 points + 2 bonus points
    - GPT and GQA (8 points)
    - Mamba and GLA (2 points)
- Model Implementation: 7 points

5 What to Submit

You need to submit a ZIP file containing the following:

- **Report**: A report in PDF with at most 8 pages. It must include your name and student ID.
- **Code**: A folder called `src` containing source code for your implementations. Your submission should include the few questions you picked from the dataset, but it must not include large datasets (e.g., GSM-8K).

[3] You only need to write either the recurrent or the parallel formulation and do not need to care about the "chunk-wise parallel" implementation of GLA.

References

[1] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[2] Joshua Ainslie, James Lee-Thorp, Michiel De Jong, Yury Zemlyanskiy, Federico Lebrón, and Sumit Sanghai. Gqa: Training generalized multi-query transformer models from multi-head checkpoints. *arXiv preprint arXiv:2305.13245*, 2023.

[3] Albert Gu and Tri Dao. Mamba: Linear-time sequence modeling with selective state spaces. *arXiv preprint arXiv:2312.00752*, 2023.

[4] Tri Dao and Albert Gu. Transformers are ssms: Generalized models and efficient algorithms through structured state space duality, 2024.

[5] Songlin Yang, Bailin Wang, Yikang Shen, Rameswar Panda, and Yoon Kim. Gated linear attention transformers with hardware-efficient training. *arXiv preprint arXiv:2312.06635*, 2023.

[6] Noam Shazeer. Glu variants improve transformer, 2020.

[7] Jiachen Zhu, Xinlei Chen, Kaiming He, Yann LeCun, and Zhuang Liu. Transformers without normalization, 2025.

[8] Mitchell Wortsman, Jaehoon Lee, Justin Gilmer, and Simon Kornblith. Replacing softmax with relu in vision transformers. *arXiv preprint arXiv:2309.08586*, 2023.

[9] Peng Lu, Ivan Kobyzev, Mehdi Rezagholizadeh, Boxing Chen, and Philippe Langlais. Regla: Refining gated linear attention. *arXiv preprint arXiv:2502.01578*, 2025.