InfoLlama Query Assistant

Jiayuan Shen
MPML
Email: jshen20@umd.edu
UID: 118464947

Qiao Qin
MPML
Email: qqin@umd.edu
UID: 120409875

Yan Liu
DATA
Email: yanliu33@umd.edu
UID: 120424595

Yuqin Feng
MPML
Email: yuqin529@umd.edu
UID: 120196935

I. Introduction

In recent years, the rapid advancement of artificial intelligence (AI) and natural language processing (NLP) has significantly transformed the landscape of human-computer interaction. Chatbots, as one of the most prominent applications of these technologies, have become integral tools in various domains, including customer service, e-commerce, healthcare, and education. Despite their widespread adoption, existing chatbot technologies still face substantial challenges in understanding context, maintaining coherent conversations, and scaling effectively to handle diverse queries from a global user base.

This project aims to develop a sophisticated chatbot using the Llama 2 model, a state-of-the-art transformer-based architecture. Llama 2 builds upon the foundational principles of the original transformer model, incorporating enhancements in layer normalization, positional encoding, attention mechanisms, and parameter efficiency[18]. These improvements enable Llama 2 to excel in natural language understanding and generation tasks, providing more accurate and contextually relevant responses.

The primary objective of this project is to build and train the Llama 2 model from scratch to create a highly effective chatbot. We will leverage the Squad dataset to train the model for generalized conversational tasks. The performance of the chatbot will be evaluated using the BLEU (Bilingual Evaluation Understudy) score, a widely recognized metric for assessing the quality of machine-generated text.

II. Background

A. History of Chatbot

The evolution of chatbots spans several decades, marked by significant advancements in technology and methodology. This section chronicles the development of chatbots from their early beginnings to the present day, highlighting key milestones.

Early Beginnings (1960s-1980s) The inception of chatbots can be traced back to the mid-1960s with the creation of ELIZA. Developed by Joseph Weizenbaum in 1966, ELIZA was the first chatbot designed to simulate a psychotherapist. It used pattern matching and substitution to create the illusion of understanding, engaging users in simple conversations[6].

Following ELIZA, Kenneth Colby introduced PARRY in 1972. PARRY simulated a person with paranoid schizophrenia, incorporating a more complex set of rules to generate responses. PARRY's ability to imitate human-like paranoia demonstrated the potential for chatbots in psychological applications[6].

Rule-Based Systems (1980s-1990s) The development of rule-based systems marked the next significant advancement. One notable example is ALICE (Artificial Linguistic Internet Computer Entity), created by Richard Wallace in 1995. ALICE utilized heuristic conversation rules and AIML (Artificial Intelligence Markup Language) to interact with users, allowing for more flexible and varied interactions compared to earlier models[6].

Statistical and Machine Learning Models (2000s) The early 2000s saw the emergence of chatbots leveraging statistical methods and machine learning techniques. SmarterChild, launched in 2001, was a precursor to modern virtual assistants like Siri. SmarterChild could carry out basic conversations and perform simple tasks, representing a significant step towards integrating chatbots into daily digital interactions[6].

Rise of AI and Deep Learning (2010s-present) The last decade has witnessed a dramatic transformation in chatbot technology, driven by advances in AI and deep learning. This era began with Apple's Siri in 2010, which used speech recognition and natural language processing to perform tasks and answer questions[6].

Following Siri, Google Now (2012) introduced context-aware recommendations, Microsoft Cortana (2014) enhanced voice command functionalities, and Amazon Alexa (2014) expanded chatbot capabilities with smart home control. The culmination of these advancements is exemplified by OpenAI's ChatGPT (2021), a large language model capable of generating highly coherent and contextually relevant text. More recently, Microsoft Copilot (2023) integrated advanced AI functionalities into Microsoft Office 365[6].

In summary, the history of chatbots reflects a continuous evolution from simple rule-based systems to sophisticated AI-driven models, leading to increasingly capable and versatile chatbots integral to various aspects of digital interaction and automation.

B. Exsisting Issues

Despite significant advancements, chatbot technology still faces several key challenges:

Contextual Understanding: Many chatbots struggle to maintain context over long conversations, leading to disjointed and less effective interactions.

Depth of Conversational Capabilities: Current chatbots often lack the ability to engage in deep, meaningful conversations, limiting their usefulness in complex scenarios.

Personalization: Personalizing responses based on individual user preferences and histories remains a significant challenge, affecting the user experience.

Scalability and Adaptability: Ensuring chatbots can handle a large number of simultaneous users while adapting to new tasks and languages is a complex and ongoing issue.

C. Motivation of Using Deep Learning

Contextual Awareness: Deep learning models, especially those using attention mechanisms or transformers, are adept at understanding the context within a conversation, which allows chatbots to maintain coherent and contextually appropriate dialogues over multiple turns of conversation.

Understandable Response: Models like GPT (Generative Pre-trained Transformer) can generate responses that are not only relevant but also indistinguishable from human text. This capability makes interactions with chatbots more natural and engaging.

Learning from Large Corpus: Deep learning models excel at processing and learning from vast amounts of data. This ability allows them to improve continuously as they are exposed to more user interactions, making them more effective over time.

Adaptability: Deep learning models can personalize conversations to individual users' styles and preferences by analyzing past interactions, enhancing user satisfaction and engagement.

Flexible: Unlike traditional machine learning, which often requires manual feature engineering, deep learning algorithms are adept at automatically discovering the representations needed for feature detection from raw data. This reduces the need for extensive preprocessing and allows for more sophisticated features that can improve the performance of the chatbot.

Generalization: The robustness of deep learning allows chatbots to handle unexpected or unknown inputs more gracefully, providing fallback responses or asking clarifying questions instead of failing outright.

III. Methodology

For this project, we intend to construct the Llama model from scratch using PyTorch and leverage the dataset on Hugging Face called SQuAD, which contains 100,000+ question-answer pairs on 500+ articles training instances that incorporate the context and question we intend to ask and the target output should be the desired text that using the information embedded within the context to answer the question. Furthermore, we intend to adopt PyTorch as our primary framework due to its scalability and functionality in performing the Llama model construction and training procedure. Finally, the BLEU score is our major metric, which has been universally adopted in the analysis of the performance of the generative model compared with the label.

A. Dataset

Stanford Question Answering Dataset (SQuAD) is a reading comprehension dataset consisting of questions posed by crowdworkers on a set of Wikipedia articles, where the answer to every question is a segment of text, or span, from the corresponding reading passage, or the question might be unanswerable[12]. There are tremendous attributes within each training instance. Still, we only care about three elements: the context, the question, and the answer, which the slide exhibits a particular training instance. We intend to create context and question the input of our model, expecting the output to be like the answer.

a Example from Dataset

Context:

> Architecturally, the school has a Catholic character. Atop the Main Building's gold dome is a golden statue of the Virgin Mary. Immediately in front of the Main Building and facing it, is a copper statue of Christ with arms upraised with the legend "Venite Ad Me Omnes". Next to the Main Building is the Basilica of the Sacred Heart. Immediately behind the basilica is the Grotto, a Marian place of prayer and reflection. It is a replica of the grotto at Lourdes, France where the Virgin Mary reputedly appeared to Saint Bernadette Soubirous in 1858. At the end of the main drive (and in a direct line that connects through 3 statues and the Gold Dome), is a simple, modern stone statue of Mary.

Question:

- To whom did the Virgin Mary allegedly appear in 1858 in Lourdes, France?

Answer:

- Saint Bernadette Soubirous.

B. Data Preprocessing

In the context of the original SQuAD dataset, we apply conventional preprocessing techniques informed by experties understanding of large language models (LLMs) to reformat the dataset for optimal compatibility with the Llama2 model architecture [7]. This process involves importing the external pre-trained tokenizer, restructuring and formatting the dataset accordingly, padding sequences within each batch, and generating the appropriate padding masks to ensure adequate and effective model training.

1) Tokenizer Configuration: We adopt the pre-trained *'google-bert/bert-base-uncased'*tokenizer, which is instrumental in transforming the vocabulary into discrete tokens. Furthermore, we integrate a total of 5 distinctive tokens to ensure the process of converting the tokens into an index correctly, avoiding the miss transformation. These specialized tokens include

- **[BOS]**: Beginning of Sentence
- **[EOS]**: End of Sentence
- **[CON]**: Commencement of Context

- **[QUE]**: Initiation of Question
- **[ANS]**: Start of Answer

Each of them designed to fulfill a specific functional role within the tokenized sequence allowing the model understand the segmentation of the input.

2) Data Formate Reconstruction: After initializing and customizing the tokenizer according to specified conditions, we structure the dataset to facilitate discrete input formatting. The input sequence is formatted as follows:

```
['[CON]'] + Context + ['[QUE]'] + Question + ['[ANS]'] + Answer
```

Moreover, the target sequence is constructed to support the auto-regressive capabilities of the model:

```
Context + ['[QUE]'] + Question + ['[ANS]'] + Answer + ['[EOS]']
```

This specific arrangement is crucial for enabling the model's auto-regressive mechanism, ensuring accurate prediction of subsequent tokens based on the preceding input sequence.

3) Generate Padding Sequence: Following the reconstruction of the dataset, we generate the padding sequence for each batch through the *collate_fn* function, wherein we customize the padding of the input and target sequences based on the length of the longest sentence within each batch and develop a padding mask. The primary aim is to standardize and thus uniformize the tensor dimensions to [Batch_size, Sequence_length]. This uniformity allows all sentences within each batch to be adjusted to the same length, facilitating the computation of the loss functions. Finally, we converted the processed tokens into integer IDs for the model to use.

C. Model Architecture

Contemporary large models often use the transformer decoder-only architecture, which is a core component of modern language models, designed to generate text by predicting the next word in a sequence [5], [17]. However, we decided to build upon the traditional transformer decoder and adopt a more advanced architecture, namely LLaMA 2. Through this project, we aim to gain a deeper understanding of the transformer decoder, LLaMA 2, and the improvements LLaMA 2 has made based on the transformer decoder[18].

Here are the main advancements that LLaMa 2 achieved:

Decoder-Only Transformers vs. LLaMa2: A Comparative Review

Normalization Techniques:

LayerNorm is well-known for its re-centering and re-scaling invariance property that has been ubiquitously adopted while training the LLMs, which maintains the same behavior during both training and inference as it normalizes across features for each example independently, leading to more consistent performance[20]. RMSNorm, which only focuses on re-scaling invariance and regularizes the summed inputs simply according to the root mean square (RMS) statistic[20]:

$$\bar{a}_i = \frac{a_i}{\mathrm{RMS}(a)} g_i, \quad \text{where} \quad \mathrm{RMS}(a) = \sqrt{\frac{1}{n}\sum_{i=1}^{n} a_i^2} \tag{1}$$

Where a_i represent the elements of the output of the attention blocks with size $[Batch\_size, Sequence\_length, d\_model]$.
Furthermore, the paper also proposes normalization before passing through the input into the attention blocks, in which we followed the implementation strategy[18].

Activation Functions in Feed-Forward Networks:

The activation function within the feed-forward layers plays a pivotal role in determining the non-linear transformation of data, allowing the model to capture the non-linearity relationships between various features and assisting the model in generating reasonable predictions[15]. The SwiGLU activation function is defined as follows[15]:

$$SwiGLU(x) = \mathrm{Swish}\,(xW1) \odot (xV) \tag{2}$$

Where the $\odot$ symbol represent the element-wise product, both $W1$ and V represent the linear layer with size $[d\_model, hidden\_dim]$ where the $hidden\_dim$ is a hyperparameter, and the Swish function can be defined as follows[15]:

$$Swish(x) = x \times \mathrm{sigmoid}(\beta x) \tag{3}$$

Where the β represents a learnable parameter, but within the context of the LLaMa2 model architecture, the author preset the learnable parameter into 1; therefore, we could leverage the advantages of the PyTorch.nn.functional library's pre-defined function *silu* to achieve the assignment[18]. The entire feed-forward process define as follows[15]:

$$FFNSwiGLU(x) = (Swish(xW1) \odot xV)\, W2 \tag{4}$$

Where $W2$ is another linear layer with shape $[hidden\_dim, d\_model]$. SwiGLU dynamically adjusts the range of activation through the sigmoid function, making the model learn intricate patterns more effectively, which allows parts of the model to selectively regulate the amount of information passing through, adapting more flexibly to different features in the data[1].

Attention Mechanisms:

The traditional decoder-transformer generally embraces the multi-head attention mechanism, which provides context for input sequence, enabling the model to understand how different words relate to each other to create meaningful sentences[19]. When multi-head attention is used, often the dimension of the **Q**, **K**, **V** matrixes share the same

shape as $[B, T, n\_heads, d\_model//n\_heads]$, where n_heads is the number of heads. Parallel attention layers are often used instead of full-dimensionality because the model can "attend to information from different representation subspaces at different positions[19]." In contrast, the LLaMa2 utilizes another analogy methodology called multi-query attention, in which the **Q** matrix remains the sample shape, whereas the **K**, **V** matrixes' dimension reduce according to different head size which generally smaller than the *n_heads* called n_kv_head[18]. Furthermore, to adapt the formulation of the scale dot product attention, the author implements the method called repeat **K**, **V**, which broadcasts the n_kv_head dimension to assure that the **Q**, **K**, **V** matrix eventually have the same shape boost the attention calculation process. This adjustment allows LLaMa2 to handle multiple aspects of information processing simultaneously, potentially enhancing its ability to manage more complex patterns or contexts. The author proposed multi-query attention - an alternative to multi-head attention with much lower memory- bandwidth requirements in the incremental setting[14].

Positional Embedding Techniques:

Incorporating positional embedding into the input data, a usable form for the models also differs. Traditional transformers use a straightforward embedding technique that maps input tokens directly into the positional embedding. LLaMa2, however, employs a distinct embedding strategy that mitigates the inherent previous positional embedding that lacks relative positional information that might optimize how input data is represented and processed internally, which could be particularly advantageous for specific types of language-based tasks[18]. The innovative method called Rotary Embedding not only includes the absolute positional information but also retains the relative positional information[8]. Within the context of LLaMa2 architecture, the author illustrates a revolutionary approach to applying the Rotary Embedding directly after reshaping the **Q** and **K** matrix into $[B, T, n\_heads, d\_model//n\_heads]$ and $[B, T, n\_kv\_head, d\_model//n\_heads]$ rather than add the absolute positional embedding before calculate the **Q** and **K** matrix like the decoder-only transformer. The Rotary Embedding calculation process can be expressed as follows[16]:

$$R^d_{\Theta,m}\mathbf{x} = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ \vdots \\ x_{d-1} \\ x_d \end{pmatrix} \odot \begin{pmatrix} \cos m\theta_1 \\ \cos m\theta_2 \\ \cos m\theta_2 \\ \cos m\theta_2 \\ \vdots \\ \cos m\theta_{d/2} \\ \cos m\theta_{d/2} \end{pmatrix} + \begin{pmatrix} -x_2 \\ x_1 \\ -x_4 \\ x_3 \\ \vdots \\ -x_d \\ x_{d-1} \end{pmatrix} \odot \begin{pmatrix} \sin m\theta_1 \\ \sin m\theta_1 \\ \sin m\theta_2 \\ \sin m\theta_2 \\ \vdots \\ \sin m\theta_{d/2} \\ \sin m\theta_{d/2} \end{pmatrix} \tag{5}$$

Where m indicates the position of the tokens and the d is equal to the d_model, but within the context of multi-query attention, $d = d\_model//n\_heads$. The parameter θ_i is a predefined parameter, which the author defined as follows[16]:

$$\theta_i = 10000^{-2(i-1)/d\_model}, i \in [1, 2, \ldots, d/2] \tag{6}$$

Rotary Positional Encoding includes relative positional information by making the angles in the rotation matrix dependent on the current word's position, and by properties of the rotation matrix, tell us how far apart two words are[8].

KV-Cache:

The KV-cache technique is an optimization used in Transformer models to improve the efficiency of autoregressive inference. It involves caching the key **K** and value **V** matrices from previous layers and reusing them in subsequent layers during the generation process. Compared to the traditional autoregressive inference, the main advantage of KV-cache is its reduced computational overhead, low latency, memory efficiency and scalability[9].

IV. Hyperparameters

The following table I indicates all the hyperparameters we used during the training process. As shown in Table I, the parameters for the model are configured to optimize performance.

TABLE I
Model configuration parameters and their values

Parameter	Value
Embedding Size	$d\_model = 2048$
Feed Forward Dimension	$hidden\_dim = 2048$
Number of Layers	$N = 12$
Number of **Q** Heads	$n\_heads = 16$
Number of **K**, **V** Heads	$n\_kv\_heads = 8$
Dropout Probability	$P_{\text{drop}} = 0.1$
Max Sequence Length	length = 1024
Vocabulary Size	vocabSize = 30592
Batch Size	8
Optimizer	AdamW
Optimizer Scheduler	CosineAnnealingLR
Learning rate	$3e-4$
Number of Epochs	60

V. Optimization

In this project, we employ two sophisticated techniques to enhance the training process: Mixed Precision Training and Flash Attention. Mixed Precision Training is designed to improve the accuracy of the model's predictions by efficiently utilizing varying numerical precisions during computation. Concurrently, Flash Attention aims to expedite the training process by optimizing the computational efficiency of attention mechanisms. These strategies are specifically tailored to both accelerate and stabilize the model's performance, thereby facilitating more efficient learning and potentially superior outcomes.

A. Mixed Precision Training

According to the original paper, a group of NVIDIA experts yielded a methodology for training deep neural networks using half-precision floating point numbers without losing model accuracy or modifying hyperparameters, which nearly halves memory requirements and, on recent GPUs, speeds up arithmetic[11].

Generally speaking, before the emergence of Mixed Precision Training, people used the IEEE 754 standard single-precision (FP32) to store all the model parameters, but due to the exponent increase of the model size and the consensual that enormous model performance better than a small model, the cost of increase memory requirements leads a novel solution that is using the half-precision (FP16) to store the weights, activations and gradients during training iterations, which significant reduce the training time and the memory requirement without at the expense of model performance[3]. The entire process can be summarized as follows[3], [13]:

- Convert weights to FP16. Initially, the neural network weights are stored in FP32 format. These are converted to FP16 for processing, while retaining a master copy in FP32. The forward pass is conducted using FP16 weights, the loss is computed, and stored in FP32 to manage large value ranges effectively and prevent overflow.
- Scale the FP32 loss to adjust the gradient magnitudes, preventing them from being too small (which can lead to ineffective gradient values during backpropagation in FP16), thus enhancing the stability of the learning process.
- Backpropagate using the scaled loss: Gradients are initially computed in FP16 to utilize computational efficiency and are immediately converted to FP32 to maintain precision and prevent errors from accumulating.
- Convert scaled gradients back to FP32 to preserve numerical stability, avoiding issues like vanishing or exploding gradients, which are more prevalent with lower precision calculations.
- Adjust the scaling on the FP32 gradients to their correct scale and update the weights using these gradients. This final adjustment ensures that the learning updates reflect the true gradient magnitudes computed from the loss.

B. Flash Attention

Computing the attention score is an extreme and intensive computational process, heavily bottlenecked by the self-attention mechanism, which has quadratic time and memory complexity[2]. Flash Attention is an attention algorithm used to reduce this problem and scale transformer-based models more efficiently, enabling faster training and inference. The central idea of the flash attention is that it considers the IO process time as a crucial factor that impedes the computational speed[4].

In the traditional attention mechanism, the matrices $\mathbf{Q}, \mathbf{K}, \mathbf{V} \in \mathbb{R}^{N \times d}$ are initially stored in the High Bandwidth Memory (HBM). The computational process unfolds in several steps:

- Matrix Loading and Dot Product Calculation: The matrices **Q** and **K** are first retrieved from the HBM. The scaled dot product of these matrices is computed, and the resulting matrix is then stored in the HBM.
- Softmax Application: This result is reloaded from the HBM and processed through the softmax function to normalize the attention scores. The output from this step is again written back to the HBM.
- Output Computation: Subsequently, the softmax output and the **V** matrix are both loaded from the HBM. The final output of the attention mechanism is computed by multiplying these matrices. This final result is stored back in the HBM before being returned as the process output.

Flash attention intends to reduce the IO process into two parts: loading the matrix and returning the result, and splitting the **Q**, **K**, **V** and intermediate matrices into smaller matrices.[4], [2] These smaller matrices can be processed entirely within the GPU's SRAM in a single pass, avoiding the communication overhead of repeatedly reading and writing them between SRAM and HBM, thereby accelerating the performance of attention operations on the GPU.

VI. Evaluation

The BLEU (Bilingual Evaluation Understudy) score is a widely used metric for evaluating the quality of machine-generated text. It compares the generated text to one or more reference texts by measuring the overlap of n-grams. The BLEU score ranges from 0 to 1, with higher scores indicating better performance.

For our chatbot, the BLEU score will help us evaluate the model's ability to generate coherent, relevant, and contextually appropriate responses. By continuously monitoring and optimizing the BLEU score during training, we aim to develop a chatbot that meets high standards of conversational quality. Table I indicates the BLEU score comparison before and after training.

TABLE II
BLEU Score Comparison Before and After Training

	Before Training	**After Training**
1-gram Precision	0.0014	0.3703
2-gram Precision	0.0000	0.1178
3-gram Precision	0.0000	0.04507
4-gram Precision	0.0000	0.01803
BLEU Score	0.0	0.1677

Even though the number does not have an impressive improvement compared with the value before training, within the rubric, while evaluating the translation task, a BLEU score that reaches 0.5 can be perceived as a high-quality machine translation within the consideration of multiple references, whereas in our cases, we merely have one reference and most of the target can be attributed as relatively short answers which might not be able to support the 3-gram identification purpose even the 4-gram. Therefore, it does indicate that our model attempts to acquire some clue about how to answer

the question in terms of the given context and question, but it cannot provide a result that is relatively accurate as we expected. Here is an example we would like to exhibit.

A Example

Context:

- Beyoncé's first solo recording was a feature on Jay Z's "'03 Bonnie & Clyde" that was released in October 2002, peaking at number four on the U.S. Billboard Hot 100 chart. Her first solo album Dangerously in Love was released on June 24, 2003, after Michelle Williams and Kelly Rowland had released their solo efforts. The album sold 317,000 copies in its first week, debuted atop the Billboard 200, and has since sold 11 million copies worldwide. The album's lead single, "Crazy in Love", featuring Jay Z, became Beyoncé's first number-one single as a solo artist in the US. The single "Baby Boy" also reached number one, and singles, "Me, Myself and I" and "Naughty Girl", both reached the top-five. The album earned Beyoncé a then record-tying five awards at the 46th Annual Grammy Awards; Best Contemporary R&B Album, Best Female R&B Vocal Performance for "Dangerously in Love 2", Best R&B Song and Best Rap/Sung Collaboration for "Crazy in Love", and Best R&B Performance by a Duo or Group with Vocals for "The Closer I Get to You" with Luther Vandross.

Question:

- How many top five singles came from her first album?

Target:

- four

Model Prediction:

- four

GPT2 Prediction:

- Beyoncé's first album was a feature on Jay Z's "'03 Bonnie & Clyde' that was

As we can observe from the context and question, the correct answer might not be easily acquired directly from the context, which entails additional understanding and review of the question carefully to answer the question.

We show the predictions of our model and gpt2, we can see that our model does have some clue about the task and makes the correct prediction. However, we also tried many other examples, and for many of them, our model doesn't generate correct answer, which suggests that our model is unable to stabilize and perform consistently.

Furthermore, due to the limitation of computational resources, we configure and train the model via the Google Colab Pro+ environment, running approximately 60 epochs. The loss generally decreased from 10 to 2. Initially, the model's predictions rely on random guessing across the extensive vocabulary of 30,592 words. However, as training progressed, the loss reduction to 2 signifies that the model has developed a significantly refined capability to select the word roughly within 10 words.

VII. Limitation

As a result, as illustrated above, we can observe that our model's performance does not achieve a state-of-art status to induce the desired answer according to the given context and question in which our model exhibits more like a pre-trained model that generates the new tokens in terms of the previous context according to the maximum likelihood issues. To demonstrate our hypothesis, we import the pre-trained GPT2 from the hugging face and execute the inference API with the same input. The result is shown in the previous section.

Therefore, in order to mitigate the problem, we might conduct further adjustments incorporating fine-tuning LLMs via Reinforcement learning from human feedback (RLHF), Supervised Fine-tuning step (SFT), Reward Modeling step (RM) to the Proximal Policy Optimization (PPO) step in which all those steps entail firm knowledge associate with reinforcement machine learning that we are currently lacking with understanding about certain concepts[10]. Furthermore, the datasets we are tackling seem to contain some Spanish and French symbols, and some of the targets might not be formatted correctly, which might impede our model due to its parameter size, which is unable to generalize and robustness to those unexpected languages.

VIII. Contribution Factor

All four of us made equal contributions to this project. Name order sequence list randomly. Each team member focuses on distinct yet complementary aspects. Specifically, Qiao Qin is dedicated to advancing our understanding of multi-query attention and the enhanced version of Rotary embedding via complex number operations associated with coding implementation and relevant sections of the project report. Meanwhile, Yan Liu is primarily responsible for assessing the dataset, implementing RMS, Rotary embedding basic approaches, KV-cache functionality in code development, and the interconnected report. Yuqin Feng is engaged in the Flash Attention and Mixed Precision Training mechanisms, integrating the relevant code into our existing framework. Lastly, Jiayuan Shen oversees data preprocessing, coding the generation process, evaluation matrices, SwiFeedForward, and the corresponding report segments.

IX. Reference

[1] Baeldung. Understanding activation functions. https://www.baeldung.com/cs/activation-functions-neural-nets, March 2024. Accessed: 2024-08-11.

[2] Tri Dao, Daniel Y. Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. Flashattention: Fast and memory-efficient exact attention with io-awareness, 2022.

[3] J. Davis. Understanding mixed precision training - towards data science. Medium, February 2024. [Online]. Available: https://towardsdatascience.com/understanding-mixed-precision-training-4b246679c7c4.

[4] Aleksa Gordić. Eli5: Flashattention - aleksa gordić - medium. Medium, July 2023. [Online]. Available: https://gordicaleksa.medium.com/eli5-flash-attention-5c44017022ad.

[5] Dirk Groeneveld, Iz Beltagy, Pete Walsh, Akshita Bhagia, Rodney Kinney, Oyvind Tafjord, Ananya Harsh Jha, Hamish Ivison, Ian Magnusson, Yizhong Wang, Shane Arora, David Atkinson, Russell Authur, Khyathi Raghavi Chandu, Arman Cohan, Jennifer Dumas, Yanai Elazar, Yuling Gu, Jack Hessel, Tushar Khot, William Merrill, Jacob Morrison, Niklas Muennighoff, Aakanksha Naik, Crystal Nam, Matthew E. Peters, Valentina Pyatkin, Abhilasha Ravichander, Dustin Schwenk, Saurabh Shah, Will Smith, Emma Strubell, Nishant Subramani, Mitchell Wortsman, Pradeep Dasigi, Nathan Lambert, Kyle Richardson, Luke Zettlemoyer, Jesse Dodge, Kyle Lo, Luca Soldaini, Noah A. Smith, and Hannaneh Hajishirzi. Olmo: Accelerating the science of language models, 2024.

[6] Ina. The history of chatbots. https://onlim.com/en/the-history-of-chatbots/#:~:text=The%20first%20chatbot%20ever%20was,created%20a%20more%20advanced%20Chatbot., February 2024. Accessed: 2024-08-11.

[7] Andrej Karpathy. Let's build gpt: from scratch, in code, spelled out, 2023. YouTube video.

[8] Ngieng Kianyew. Understanding rotary positional encoding - ngieng kianyew - medium. Medium, July 2024. [Online]. Available: https://medium.com/@ngiengkianyew/understanding-rotary-positional-encoding-40635a4d078e.

[9] João Lages. Transformers kv caching explained - joão lages - medium. Medium, October 2023. [Online]. Available: https://medium.com/@joaolages/kv-caching-explained-276520203249.

[10] Nathan Lambert. Rlhf - hugging face deep rl course. https://huggingface.co/learn/deep-rl-course/en/unitbonus3/rlhf. Accessed on August 11, 2024, Eastern Time.

[11] Paulius Micikevicius, Sharan Narang, Jonah Alben, Gregory Diamos, Erich Elsen, David Garcia, Boris Ginsburg, Michael Houston, Oleksii Kuchaiev, Ganesh Venkatesh, and Hao Wu. Mixed precision training, 2018.

[12] Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. SQuAD: 100,000+ questions for machine comprehension of text. In Jian Su, Kevin Duh, and Xavier Carreras, editors, *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 2383–2392, Austin, Texas, November 2016. Association for Computational Linguistics.

[13] Sebastian Raschka. Accelerating large language models with mixed-precision techniques. Lightning AI, June 2023.

[14] Noam Shazeer. Fast transformer decoding: One write-head is all you need, 2019.

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

[16] Jianlin Su, Yu Lu, Shengfeng Pan, Ahmed Murtadha, Bo Wen, and Yunfeng Liu. Roformer: Enhanced transformer with rotary position embedding, 2023.

[17] Gemma Team, Thomas Mesnard, Cassidy Hardin, Robert Dadashi, Surya Bhupatiraju, Shreya Pathak, Laurent Sifre, Morgane Rivière, Mihir Sanjay Kale, Juliette Love, Pouya Tafti, Léonard Hussenot, Pier Giuseppe Sessa, Aakanksha Chowdhery, Adam Roberts, Aditya Barua, Alex Botev, Alex Castro-Ros, Ambrose Slone, Amélie Héliou, Andrea Tacchetti, Anna Bulanova, Antonia Paterson, Beth Tsai, Bobak Shahriari, Charline Le Lan, Christopher A. Choquette-Choo, Clément Crepy, Daniel Cer, Daphne Ippolito, David Reid, Elena Buchatskaya, Eric Ni, Eric Noland, Geng Yan, George Tucker, George-Christian Muraru, Grigory Rozhdestvenskiy, Henryk Michalewski, Ian Tenney, Ivan Grishchenko, Jacob Austin, James Keeling, Jane Labanowski, Jean-Baptiste Lespiau, Jeff Stanway, Jenny Brennan, Jeremy Chen, Johan Ferret, Justin Chiu, Justin Mao-Jones, Katherine Lee, Kathy Yu, Katie Millican, Lars Lowe Sjoesund, Lisa Lee, Lucas Dixon, Machel Reid, Maciej Mikuła, Mateo Wirth, Michael Sharman, Nikolai Chinaev, Nithum Thain, Olivier Bachem, Oscar Chang, Oscar Wahltinez, Paige Bailey, Paul Michel, Petko Yotov, Rahma Chaabouni, Ramona Comanescu, Reena Jana, Rohan Anil, Ross McIlroy, Ruibo Liu, Ryan Mullins, Samuel L Smith, Sebastian Borgeaud, Sertan Girgin, Sholto Douglas, Shree Pandya, Siamak Shakeri, Soham De, Ted Klimenko, Tom Hennigan, Vlad Feinberg, Wojciech Stokowiec, Yu hui Chen, Zafarali Ahmed, Zhitao Gong, Tris Warkentin, Ludovic Peran, Minh Giang, Clément Farabet, Oriol Vinyals, Jeff Dean, Koray Kavukcuoglu, Demis Hassabis, Zoubin Ghahramani, Douglas Eck, Joelle Barral, Fernando Pereira, Eli Collins, Armand Joulin, Noah Fiedel, Evan Senter, Alek Andreev, and Kathleen Kenealy. Gemma: Open models based on gemini research and technology, 2024.

[18] Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, Dan Bikel, Lukas Blecher, Cristian Canton Ferrer, Moya Chen, Guillem Cucurull, David Esiobu, Jude Fernandes, Jeremy Fu, Wenyin Fu, Brian Fuller, Cynthia Gao, Vedanuj Goswami, Naman Goyal, Anthony Hartshorn, Saghar Hosseini, Rui Hou, Hakan Inan, Marcin Kardas, Viktor Kerkez, Madian Khabsa, Isabel Kloumann, Artem Korenev, Punit Singh Koura, Marie-Anne Lachaux, Thibaut Lavril, Jenya Lee, Diana Liskovich, Yinghai Lu, Yuning Mao, Xavier Martinet, Todor Mihaylov, Pushkar Mishra, Igor Molybog, Yixin Nie, Andrew Poulton, Jeremy Reizenstein, Rashi Rungta, Kalyan Saladi, Alan Schelten, Ruan Silva, Eric Michael Smith, Ranjan Subramanian, Xiaoqing Ellen Tan, Binh Tang, Ross Taylor, Adina Williams, Jian Xiang Kuan, Puxin Xu, Zheng Yan, Iliyan Zarov, Yuchen Zhang, Angela Fan, Melanie Kambadur, Sharan Narang, Aurelien Rodriguez, Robert Stojnic, Sergey Edunov, and Thomas Scialom. Llama 2: Open foundation and fine-tuned chat models, 2023.

[19] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2023.

[20] Biao Zhang and Rico Sennrich. Root mean square layer normalization, 2019.