PretrainedLLM Report

Entang Wang
enwa00001@stud.uni-saarland.de

Zhenyu Feng
zhfe00001@stud.uni-saarland.de

Abstract

Pretrained large language models (LLMs) have demonstrated remarkable capabilities, yet the detailed pretraining process remains poorly documented and is often inaccessible due to computational demands. This report presents our experiences pretraining a compact GPT-2 class model with the goal of enhancing downstream task performance solely through carefully planned modifications to model architecture and pretraining procedures. Operating under tight computational and temporal constraints, we systematically evaluated hyperparameter choices, such as employing learning rate schedulers, extending context windows, and replacing conventional post-layer normalization with pre-layer normalization. Our analyses provide a basic insight into the influence of these changes on validation tasks, highlighting the trade-offs involved and accumulating some practical experience for future study in resource-constrained LLM pretraining.

1 Introduction

Pretrained large language models (LLMs) have revolutionized natural language processing (NLP) by significantly advancing performance across a wide array of tasks. However, the complexity and scale of state-of-the-art (SoTA) LLMs often obscure the practical details of their pretraining process, limiting opportunities for hands-on exploration and experimentation for many researchers and students.

In this study, we address these limitations by providing an accessible yet rigorous exploration of LLM pretraining using a smaller GPT-2-based architecture. Our objective is to incrementally optimize the model's performance exclusively through targeted enhancements of the pretraining strategy and architectural adjustments. Specifically, we explore the effects of several key modifications: employing learning rate schedulers to stabilize training, extending the model's context length to capture broader textual information, and replacing traditional post-layer normalization with pre-layer normalization (pre-layernorm), as recent literature suggests this enhances training stability and performance.

Throughout this project, we worked under strict resource constraints, necessitating precise and analytical experimentation to achieve noticeable improvements without exhaustive hyperparameter searches. Our approach emphasizes systematic experimentation and data-driven analysis to understand how each modification influences downstream task performance. By transparently documenting our process and outcomes, this report contributes a practical case study that elucidates the nuances of LLM pretraining and provides valuable insights for future efforts operating under similar constraints.

The remainder of this report details our methodology, experimental setup, results, analyses, and final conclusions.

2 Phase 1: Baseline Model and Pretraining

2.1 Model Architecture

For our baseline model, we selected a GPT-2-style Transformer architecture(**?**) which is a decoder-only architecture with attention mechanism.

Specifically, our base model consists of 6 layers Transformer block and each block has 6 attention heads. Other model paramaters are shown in 1.

We chose the GPT-2 architecture as the baseline due to its proven generative ability and excellent performance across a variety of NLP tasks. It is suitable for pretraining and can serve as a strong foundation for further downstream tasks.

2.2 Dataset and Validation Tasks

We used a self-customized BookCorpus(**?**) dataset for pretraining, which contains extensive, diverse

Model Parameter	Value
Vocabulary Size	50257
Embedding Size	768
Layer Number	6
Head Number	6
FeedForward Layer Dimension	3072
Embedding Dropout	0.1
Attention Dropout	0.1
Residual Dropout	0.1
Feedforward Dropout	0.1
Max Context Length	512

Table 1: Base Model Parameter Settings

English text derived from thousands of unpublished books. The dataset contains not only the original content in BookCorpus but also mixes up some common knowledge collected from Wikipedia, like "What are the three primary colors?". The dataset was preprocessed by concatenating paragraphs into longer text chunks, then tokenizing these chunks with a maximum sequence length of 512 tokens. Shorter sequences were padded using the GPT-2 tokenizer's EOS token (token-id: 50256).

2.3 Training Process

Our hyperparameters were selected based on commonly used best practices and empirical experimentation:

Hyperparameter	Value
Batch Size	16
Learning Rate	2.5e-4
Optimizer	AdamW $\beta(0.9, 0.95)$
Weight Decay	0.01

Table 2: Base Model Hyperparameter Settings

These hyperparameters were informed by recommendations from literature and previous successful pretraining experiments with transformer-based models(**?**).

Our pertaining process applies self-regression, where the model is trained in an autoregressive manner. In this framework, the input sequence is shifted by one position to serve as the target, and the model learns to predict the next token based solely on the preceding context.

During training, the cross-entropy loss is computed over the predicted tokens while ignoring the loss contribution from padding tokens (e.g., the EOS token). We update model parameters using the AdamW optimizer, and mixed precision training (enabled via PyTorch AMP) is utilized to accelerate computation and optimize memory usage.

Figure 1: Base Model Average Loss

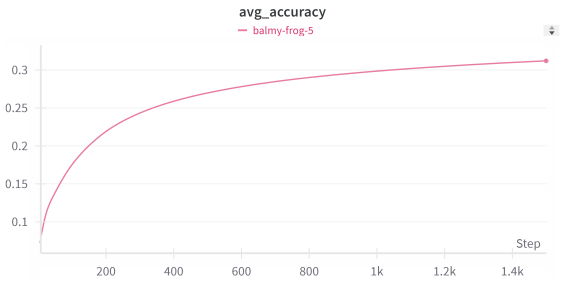

Figure 2: Base Model Average Accuracy

Figure 3: Base Model Batch Loss

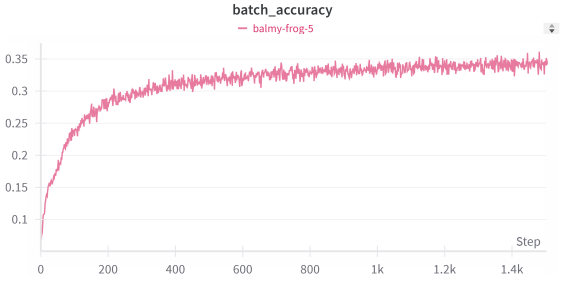

Figure 4: Base Model Batch Accuracy

2.4 Evaluation

To ensure consistent progress monitoring and facilitate debugging, we integrate real-time logging with wandb. Training metrics such as batch loss and token-level accuracy are logged at regular intervals, and model checkpoints are saved periodically (every 10% of total training batches) for later evaluation and potential recovery. This comprehensive approach allows us to efficiently pretrain the baseline transformer model while closely monitoring its performance on validation tasks. The final base model average loss converged at 3.76 while the final base model average accuracy converged at 31.2%.

The initial pretraining run over our customized BookCorpus dataset showed a weak ability:

2.4.1 Perplexity Test

To ensure that perplexity reflects the entire dataset performance and is unaffected by batch size variations, we compute the mean perplexity by summing the token-level loss across all batches, averaging it over the total number of tokens in the dataset, and then applying the exponential function to obtain the final perplexity.

The test datasets we used are as follows:

- Openwebtext-10k (**?**)
- Bookcorpus-5k (first 5k samples)
- Medical-o1-reasoning-SFT (whole Question part) (**?**)

Test Dataset	PPL (↓)
openwebtext-10k	13931
bookcorpus-5k	1.322
medical-o1-reasoning-SFT-Question	4.257

Table 3: Base Model Perplexity Test

The outcome results are shown in table 3.

2.4.2 Text Generation

We utilize the end-of-text token as the prompt to facilitate the generation of completely free-form text. This approach minimizes the constraints typically introduced by predefined prompts and mitigates any strong directional influence on the model's output, allowing for more diverse and unbiased text generation. To encourage more creative and less repetitive outputs, we set the temperature to 0.8, which increases randomness in token selection, and apply top-k sampling with k=40. Additionally, we generate up to 200 new tokens per sentence and repeat this process for 10 times to ensure a varied and expressive set of outputs. For text evaluation, we provide the generated sentences to GPT-4o, which rates them based on four criteria: (1) Grammar and Syntax, (2) Semantic Clarity, (3) Contextual Relevance, and (4) Creativity and Style. Each criterion is scored on a scale of 1 to 10. The scores for the 10 sentences are averaged for each metric, and the final rating is obtained by summing the four averaged scores. Therefore, the maximum possible score for a model is 40.

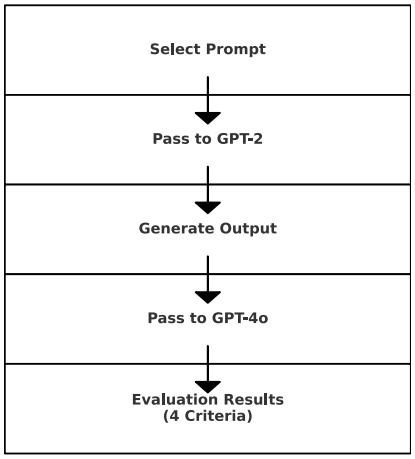

Figure 5: Process Flowchart for Text Generation.

The prompt we used for GPT-4o is like: You are a language expert tasked with evaluating a set of generated sentences. Please rate each sentence based on the following criteria: Grammar and Syntax (1-10): Does the sentence follow proper grammar and syntax rules? Semantic Clarity (1-10): Is the meaning of the sentence clear and easy to understand? Contextual Relevance (1-10): Is the sentence relevant to each other? Creativity and Style (1-10): Does the sentence demonstrate creativity or an appropriate style? For each sentence, provide a detailed score (1-10) for each category.

Table 5 is an example text generated by our base model.

2.4.3 QA Test

We prepare 281 QA pairs with short answers. Each QA pair follows the format:

"question": "What is the largest planet?"

Criteria	Quality Value(↑)
Grammar and Syntax	3.9
Semantic Clarity	4.7
Contextual Relevance	4.2
Creativity and Style	4.7

Table 4: Base Model Text Generation Test

you don't have to be kidding me .i 'm serious ! "i don't want to be .it 's just for you and that 's all . "i do not .i don't want to be this close to you. she 's a smart girl . "i am not perfect .i 'm not per -fect ."i don't care what you think . "i 'm sure you didn't see each other when you were a child . "how do you feel about it ?

Table 5: An Example of Base Model Text Generation

"answer": "Jupiter"

The question part is used as the input prompt to predict the next token. The negative log-likelihood of the first answer token, in this case, 'J', is stored and then appended to the question as a hint. This updated prompt is then used to generate the negative log-likelihood of the next token, 'upiter'. This iterative process continues until the full answer is generated. The final score for a QA pair is obtained by summing the negative log-likelihood of all answer tokens.

Calculation for the example above:

$-\log(P($ "J" | "What is the largest planet? "$))$ = 15.19

$-\log(P($ "upiter" | "What is the largest planet? J" $))$ = 12.65

Therefore, the score for this QA pair is 15.19 + 12.65 = 27.84.

To evaluate model performance, we compute the overall score by averaging across 228 QA pairs. For the base model, the overall averaged **QA score** is 29.78.

3 Phase 2: Initial Analysis

From the above experiment results, we can see our base model which is limited by the parameters and training data scale has learned certain linguistic patterns from the BookCorpus data. However, the performance on PPL evaluation tasks suggests that it struggles to generalize beyond its primary training domain. In text generation task, While the generated sentences maintain basic grammatical structure, the overall semantic clarity and contextual relevance are suboptimal. The generated text often exhibits abrupt topic shifts and occasional logical inconsistencies. The QA test evaluation indicates that the base model finds it struggling to align question-context with precise token-level predictions.

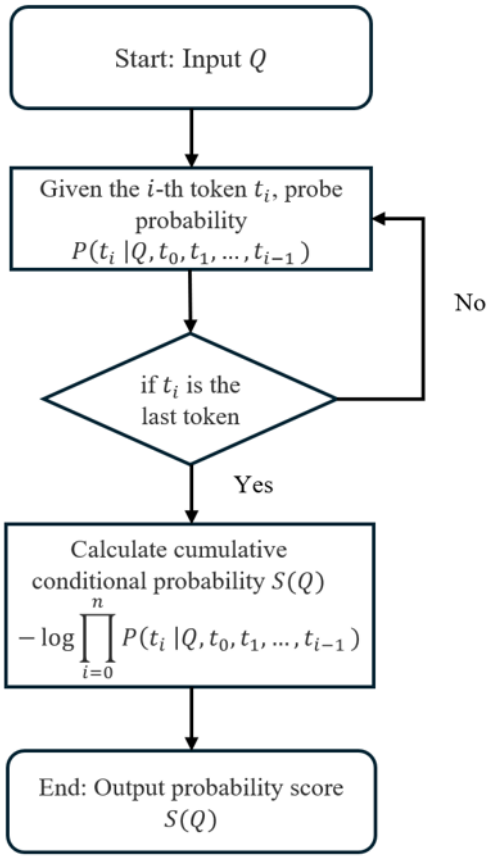

Figure 6: Process Flowchart for one QA Test.

Overall, this performance is likely attributable to insufficient parameter capacity and restricted exposure to suffient and diverse training data.

4 Phase 3: Iterative Improvements

4.1 Increase Model Parameters

According to the scaling law (**?**), model performance is generally positively correlated with both the volume of training data and the number of model parameters. In this phase, we enhance the model's capacity by increasing the number of transformer blocks and attention heads. Specifically, our updated model comprises 12 transformer block layers, each equipped with 12 attention heads. Our design choices are further motivated by the previ-

ous research findings that suggest larger models are capable of better generalization and few-shot learning performance (**?**). Thus, we got a new model–`GPT2-para`.

Model Parameter	Value
Vocabulary Size	50257
Embedding Size	768
Layer Number	12
Head Number	12
FeedForward Layer Dimension	3072
Embedding Dropout	0.1
Attention Dropout	0.1
Residual Dropout	0.1
Feedforward Dropout	0.1
Max Context Length	512

Table 6: Updated Model Parameter Settings

Hyperparameter	Value
Batch Size	4
Learning Rate	2.5e-4
Optimizer	AdamW $\beta(0.9, 0.95)$
Weight Decay	0.01

Table 7: Updated Model Hyperparameter Settings

4.2 Extend Training Data

As we discussed in the former section, data quantity is a critical factor in pretraining transformer-based models. In this phase, we double the size of the training dataset compared to that used in the baseline model. The enlarged dataset not only provides a richer variety of linguistic patterns but also facilitates improved model generalization across diverse tasks. By incorporating additional data, the model is exposed to a broader range of contexts and differences, which is expected to yield more diverse representations and enhanced downstream performance.

According to the training curve 7, 8, 9, 10 (the **yellow** line refers to the base model architecture trained with extended dataset while the **orange** line refers to the updated model trained with extended dataset), we can see after increasing the model parameters and the data quantity, compared to the base model trained with extended dataset whose batch loss converged at 3.3 and the batch prediction accuracy converged at 35%, the updated model can converge at a lower batch loss (3.0) and achieve a higher batch prediction accuracy (39%). During the training process, the updated model's performance is continuously better than the base model. Through this, we got model–`GPT2-data`.

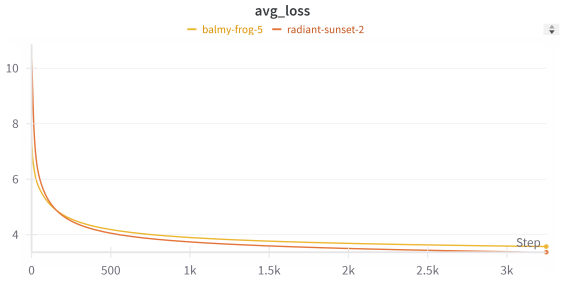

Figure 7: Model Average Loss Comparison

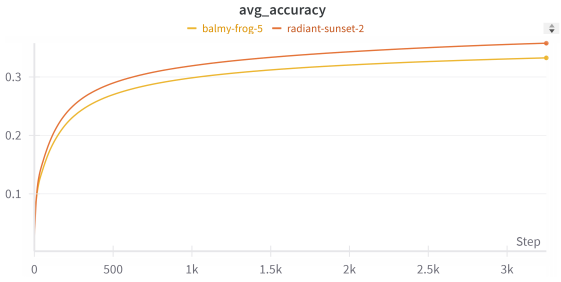

Figure 8: Model Average Accuracy Comparison

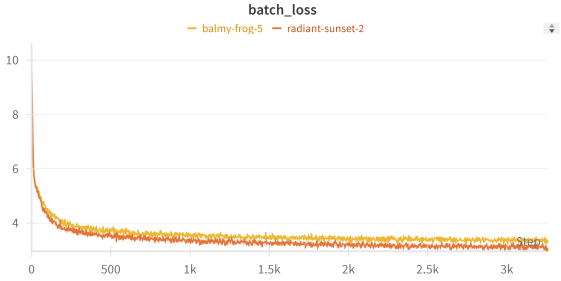

Figure 9: Model Batch Loss Comparison

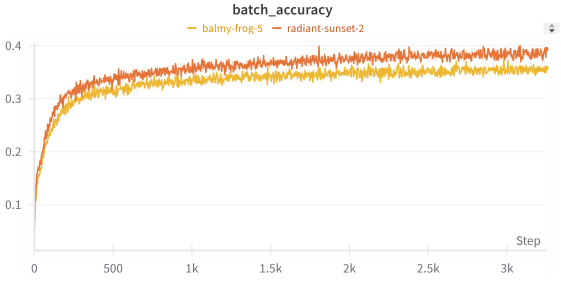

Figure 10: Model Batch Accuracy Comparison

4.3 Use Learning Rate Scheduler

Our preliminary training experiments revealed that maintaining a fixed learning rate in the latter stages of training often results in oscillatory behavior of the loss, preventing a smooth convergence. To address this issue, we employed a dynamic learning rate scheduler, specifically a cosine annealing scheduler with a warm-up phase. This approach allows the learning rate to gradually increase during the initial training period, thereby stabilizing the early learning process, and then decay smoothly to facilitate fine-grained convergence in later stages. Such dynamic adjustment has been demonstrated to improve optimization and reduce training instability in deep neural networks (**?**). We got the model–`GPT2-lr` from this.

Item	Value
Initial Learning Rate	6e-4
Scheduler Type	Cosine
Warm Up Percent	3%
Traning Steps	Batches Number

Table 8: Learning Rate Scheduler Settings

4.4 Improve Data Quality

The quality of the training dataset is a crucial determinant of model performance. In this section, we focused on refining the base dataset (BookCorpus) to enhance data quality. First, we preprocessed the text by omitting repetitive termination tokens, such as newline characters (\n), tab characters (\t), and extraneous whitespace, which can introduce noise during training. Furthermore, we extended the effective capacity of the dataset by including data chunks that are longer than 0.8 times the model's context length; these chunks are padded at the end with the EOS token to reach the desired length. During training, the loss computation explicitly ignores these padding tokens, thereby ensuring that they do not contribute to gradient updates. Additionally, we augmented the dataset with manually crafted instructions to help the model learn structured output generation for common queries, thereby reinforcing its capability in instruction-following tasks. We generate a model–`GPT2-instruct`.

4.5 Modify Model Structure

Beyond scaling parameters and data augmentation, structural modifications to the model architecture can also yield performance gains. In this section, we explored model structure refinements. As an example, we experimented with replacing the pre-layer normalization setup with the conventional post-layer normalization configuration. This pre-layer normalization has been adopted in many contemporary GPT-class models. Pre-layer normalization can lead to more stable training and improved gradient flow, especially in deep transformer architectures. We trained a model–`GPT2-Post`.

4.6 Results

4.6.1 Perplexity Test

To ensure fair comparisons across datasets with different scales, we applied max normalization to each dataset and then average them to get a balanced evaluation of model performance across domains.

Model	Openweb	Bookcorpus	Medical	Average(↓)
`Base Model`	1.0	1.0	1.0	1.0
`GPT2_para`	1.0	0.997	0.937	0.978
`GPT2_data`	0.987	0.985	0.924	0.966
`GPT2_lr`	**0.484**	**0.976**	**0.864**	**0.774**
`GPT2_instruct`	0.907	0.984	1.0	0.963
`GPT2_Post`	0.695	1.0	0.902	0.866

Table 9: Different Models Perplexity Test Results

4.6.2 Text Generation

Model	Text Quality (Maximum: 40)(↑)
`Base Model`	17.5
`GPT2_para`	18.6
`GPT2_data`	**19.1**
`GPT2_lr`	17.5
`GPT2_instruct`	9.2
`GPT2_Post`	18.5

Table 10: Different Models Text Generation Test Results

We show some text generation examples in the appendix.

4.6.3 QA Test

The QA test results are as table 11 shows.

5 Phase 4: Further Analysis

Below, we provide a more in-depth discussion of what worked, what did not, and possible reasons behind these outcomes.

- Scaling Model Parameters and Dataset

Model	QA Score(↓)
Base Model	29.78
GPT2_para	28.37
GPT2_data	28.83
GPT2_lr	30.41
GPT2_instruct	**23.85**
GPT2_Post	30.02

Table 11: Different Models QA Test Results

GPT2_para and GPT2_data demonstrate the importance of parameter count and data quantity. Increasing the number of transformer layers and heads (GPT2_para) improved QA performance and slightly lowered perplexity. Similarly, adding more training data (GPT2_data) provided a notable boost in text generation quality. However, the perplexity on OpenWebText still remained relatively high. This suggests that while more data and parameters generally help, domain diversity and data quality are equally critical for broader generalization.

- Learning Rate Scheduler

  GPT2_lr yielded a significant improvement in perplexity, indicating that dynamic learning rates can stabilize later-stage optimization and help the model converge to a better solution. However, text generation quality did not improve, and QA performance even decreased. One plausible explanation is that while the scheduler facilitates better next-token prediction (as evidenced by perplexity), it does not necessarily align the model's internal representations with more coherent, contextually relevant or diverse content. Moreover, the QA task might require more specialized context understanding or instruction-like data, which was not directly addressed by the learning rate scheduler.

- Improve Data Quality

  GPT2_instruct stands out in QA tasks, achieving a substantially lower average QA score than the baseline. This improvement underscores how instruction-like data or question-answer pairs can directly enhance the model's ability to predict correct tokens for factual queries. As a side effect, the text generation score decreased, suggesting that heavily instruction-oriented training data may reduce the model's creative flexibility when generating open-ended text. The model might overfit to the instruction-following style, resulting in less diverse or more "template-like" generation outputs.

- Post Layer Normalization

  GPT2_Post exhibits only modest improvement in perplexity over the baseline model. However, its performance remains noticeably inferior compared to models that adopt pre-layer normalization. This outcome suggests that using a post-layer normalization configuration may actually weaken the model's overall efficacy, underscoring the advantage of pre-layernorm setups in achieving better training stability and language generation quality.

5.1 Discussion

Our experimental results reveal that each modification impacts the model in distinct ways. For example, incorporating instruction-like data markedly improves QA performance, yet it comes at the cost of deteriorating free-form text generation. Similarly, while the learning rate scheduler effectively reduces perplexity, this improvement does not consistently translate into enhanced open-ended generation or QA outcomes.

It is important to note that our study is subject to significant limitations. Due to constrained computational resources and a limited dataset, the model scales employed remain relatively small. As a result, the models may not have reached the threshold where emergent abilities of LLM would happen.

Overall, our results highlight the trade-offs inherent in each modification strategy under resource-constrained conditions, systematically and scientifically improving model performance under a constrained compute budget.

how are we going to get him to agree that we should get him off this planet ? i mean , he 'd have to go back to human form , but how ? "“ i don't know. but he 's going to have to go back to human form. there 's no way we can do that . (GPT_512_100_2)

it was only for a moment , but he did n't release a shudder .his body was tense , ready to go on , but he had n't thought , not quite yet.but something about him was getting closer , closer , closer, closer ... closer (GPT_Base_512_100)

with no idea what was happening .it was all very fascinating. i had to stop thinking about what was happening to me .but i just couldn't stop .she 'd been right .she had to stop being right .it wasn't exactly easy. i went to the bathroom , and it was empty. i stood outside , and my mind was in a state of shock. (GPT_512_50_2)

the given sentence into the correct form:
The sky is a deep blue.
The sky is a deep blue.
The sky is a deep blue.
The sky is a deep blue.
The sky is a deep blue.
The stars in the sky are red.
The sky is a deep blue.
The sky is an awe-inspiring blue.
There is a wide, deep blue,
which is an awe-inspiring blue.
(GPT_Alpaca_512_100)

and the blood drained from his mouth .he could n't breathe .he had to get a hold of her , if that meant she was alive .his lips touched her forehead and she opened her mouth to speak , but he did n't stop there. he did n't even have to move .he looked up at her , his eyes filled with concern .she was standing there , staring at him(GPT_Base_512_100)

Table 12: Generated Text Examples of 5 Models