

Fine-Tuning GPT-2 for Sentiment Analysis, Paraphrase Detection, Sonnet Generation and Political Affiliation Detection

Stanford CS224N {Default} Project

Samantha Leventis, Iris Xu, Anna Wu

Department of Computer Science

Stanford University

sammy7@stanford.edu, izxu28@stanford.edu, wuanna@stanford.edu

Abstract

Large language models such as GPT-2 have demonstrated strong performance across a wide range of natural language tasks, but adapting them effectively to specialized downstream tasks remains challenging. Fine-tuning requires balancing general linguistic knowledge with task-specific learning while maintaining stable optimization. Additionally, different tasks, such as classification and text generation, have distinct requirements, making it difficult for a single framework to perform well across them.

In this work, we implement GPT-2 and fine-tune it using several optimization and training strategies, including the Adam and Shampoo optimizers, LoRA PEFT, and Direct Preference Optimization (DPO). We conduct an ablation study evaluating different combinations of these techniques while fine-tuning GPT-2 on four specialized tasks: sentiment analysis, paraphrase detection, sonnet generation, and political affiliation detection. Our models outperform the provided baselines across all tasks, demonstrating that improved optimization strategies and fine-tuning methods can substantially improve the performance of GPT-2 across diverse NLP tasks. We also observe that the benefits of these techniques are task-dependent, with some methods proving better suited to particular problem settings than others.

TA Mentor: Sarah Chen

External Collaborators or Mentor: No

Sharing Project: No

Team member	Late days remaining
Samantha Leventis	6
Iris Xu	5
Anna Wu	5
Total pooled	16
Team size	3
Late days applied for the report	3

1 Introduction

Large language models (LLMs) have dramatically improved the ability of machine learning systems to perform a wide range of tasks. By training on large-scale web data, these models learn general linguistic representations that enable strong zero-shot performance across tasks including question answering, translation, and text generation. However, despite these advances, significant challenges remain in adapting large pretrained language models to successfully and accurately complete downstream tasks, which often require models to learn task-specific representations that differ substantially in structure and objective. For example, classification tasks such as sentiment analysis or political affiliation detection require the model to aggregate information across an input sequence, whereas generative tasks such as poetry generation require the model to produce coherent text while maintaining stylistic constraints. Successfully adapting a single pretrained model across such diverse objectives requires further research into best optimization strategies, fine-tuning methodologies, and decoding techniques.

In this work, we explore how different fine-tuning and optimization strategies affect the ability of GPT-2 to adapt to multiple natural language processing tasks. We implement GPT-2 and investigate the efficacy of various fine-tuning techniques for improving its performance on four downstream tasks: sentiment analysis, paraphrase detection, sonnet generation, and political affiliation detection.

The **sentiment analysis** task requires the model to classify the emotional polarity of a movie review. Using the Stanford Sentiment Treebank (SST) dataset, the model takes a review as input and predicts one of five sentiment labels: positive, somewhat positive, neutral, somewhat negative, or negative. For the CFIMDB dataset, the model performs a binary classification task, labeling reviews as either positive or negative. The **paraphrase detection** task evaluates whether a pair of questions from the Quora dataset conveys the same meaning, i.e., whether one second question is a paraphrase of the other. The **sonnet generation** task asks the model to produce a stylistically coherent continuation of three initial lines from a Shakespearean sonnet. Finally, **political affiliation detection** (our custom task) requires the model to predict the political orientation of an input text statement. The model outputs a label from 0 to 4, where 0 represents the most conservative stance and 4 represents the most liberal.

For each task, we implement the fine-tuning forward function, integrate extensions—namely, LoRA PEFT, the Shampoo optimizer and DPO—and experiment with full-model and last-layer fine-tuning, hyperparameters, and various decode-time strategies for text generation. We demonstrate that simple fine-tuning can substantially improve the adaptability of pretrained language models across heterogeneous NLP tasks and provide analyses of the applicability of various fine-tuning methods to different tasks.

2 Related Work

Large language models have demonstrated the ability to generalize across a wide range of tasks without task-specific supervision. In particular, the work "Language Models are Unsupervised Multitask Learners" [1] introduced GPT-2 and showed that sufficiently large language models trained on broad web text corpora can perform downstream tasks such as translation, question answering and summarization in a zero-shot setting. This finding suggested that language modeling alone can produce representations that are useful for many natural language processing tasks. Earlier work exploring emergent capabilities in sequence models includes "Generating Wikipedia by Summarizing Long Sequences" [2], which demonstrated that models trained to summarize long documents were able to implicitly learn related linguistic tasks, such as translating named entities across languages. These results helped motivate the hypothesis that training on large-scale reading and summarization tasks may induce general language understanding capabilities. Our work builds on this idea by applying language models to downstream tasks such as classification of text, creative writing, and our custom task, political affiliation detection.

We also consulted the following papers for our extensions:

1. **Shampoo** [3] is a second-order preconditioning method that approximates curvature information using matrix statistics of gradients. Compared with first-order adaptive optimizers such as Adam, Shampoo can achieve improved convergence by better conditioning updates along different parameter dimensions. In our project, we implement a Shampoo optimizer from scratch based on the paper.
2. **FlashAttention** [4] introduces an IO-aware attention algorithm that reduces memory usage and improves training speed by computing attention in tiled blocks directly in GPU memory. We do not implement FlashAttention from scratch, but rather experiment with using the existing PyTorch implementation of dot-product attention accelerated with FlashAttention.
3. **LoRA** [5] is a method that injects trainable low-rank matrices into the attention layers of a pretrained transformer while freezing the original pretrained model weights. This significantly reduces the number of trainable parameters required for downstream tasks while preserving most of the model’s performance. We implement a LoRA-based attention class from scratch based on the paper.
4. **Direct Preference Optimization (DPO)** [6] is a preference-based fine-tuning approach designed to align language models with human preferences without explicitly training a reward model or performing reinforcement learning. Specifically, DPO fine-tunes a model on a dataset where each entry consists of a prompt and two candidate responses, one preferred and one rejected. The objective increases the likelihood of preferred responses relative to rejected responses while regularizing the policy to remain close to the reference model. In our project, we implement DPO from scratch based on the paper.

3 Approach

3.1 Tasks

Sentiment Classification: We implement two versions of the forward function for classification tasks. In our preliminary implementation, the forward function calls GPT-2, retrieves the last output token, and passes it through a linear layer with optional dropout. The linear layer computes

$$y = xA^T + b,$$

where x is the hidden state of the last token and A, b are learned parameters. This version serves as a simple baseline.

Based on the observation that using only the last token discards most of the sequence information, we replace the last-token representation with a strategy incorporating mean pooling over all token embeddings, masking out padding tokens. Dropout is applied after pooling and before the linear layer.

Paraphrase Detection: We implement the forward function, which calls the GPT-2 model, retrieves the last output token and calls the `hidden_state_to_token()` layer to produce output logits for each class, either 0.0 or 1.0. We did not use the provided `paraphrase_detection_head` as it did not produce logits in the correct format. We believe this was a bug that was meant to be identified and fixed by us.

Sonnet Generation: We implement the forward function, which retrieves the last hidden state of the model. We also introduce the following modifications:

- **Freezing:** In theory, the first layers of our GPT-2 model are used to understand fundamental characteristics of language, such as grammar. When training for a higher-level style-related objective like sonnet generation, these characteristics should remain mostly intact. We experiment with freezing all but the last n layers (except for the layer norms, which are always unfrozen to allow a bit of flexibility).
- **Repetition Penalty:** Inspired by the Hugging Face Transformers `generate()` method [7], we optionally enforce a 1.5x repetition penalty for previously encountered tokens, or a strict ban (directly setting logits to -inf) for n-grams that have been seen before, to prevent the model from falling into repetition loops or trying to achieve higher scores by directly repeating phrases from the prompt.
- **Beam Search:** We implement beam search to seek ahead during generation to identify the most probable sequences. Since exact search over all possible sequences is computationally intractable due to the exponential size of the output space, beam search approximates this optimization by maintaining a fixed-size set of the top- k partial hypotheses at each decoding step, where k is the beam's "width." At step t , each partial sequence in the beam is expanded with all possible next tokens, producing candidate sequences. The score of a candidate sequence $y_{1:t}$ is computed using cumulative log-probability. Among all expanded candidates, only the top- k sequences with the highest scores are retained for the next step. This process continues until an end-of-sequence token is generated or a maximum sequence length is reached. To mitigate the bias toward shorter sequences introduced by log-probability accumulation, a length normalization term is also applied:

$$s_{\text{norm}}(y_{1:t}) = \frac{1}{t^\alpha} \sum_{i=1}^t \log \pi_\theta(y_i | x, y_{<i})$$

where α is a hyperparameter controlling the strength of the length normalization.

- **Top-k Filtering:** We force the next token to be sampled from the k most probable next tokens to filter out nonsense words.
- **Adaptive Learning Rate:** We dynamically halve the learning rate when loss fails to improve in order to more carefully approach minima. We also test the already-implemented cosine annealing scheduling for dynamically adjusting the learning rate [8].
- **Creating Data:** Since the Shakespeare sonnet dataset is very small, we use GPT-5.4 to generate more sonnets and extend the dataset to 1000 sonnets.
- **CHRF Stopping:** We implement patience and early stopping, but substitute the traditional cross-entropy-loss-based validation with CHRF on a 0.02 split of the dataset in order to provide a more realistic prediction of the model's test performance. CHRF uses the following formula:

$$CHRF\beta = (1 + \beta^2) \frac{CHRP * CHRR}{\beta^2 * CHRP + CHRR}$$

[9] where CHRP and CHRR are, respectively, the average precision and recall and β is a parameter weighting their relative importance.

- **DPO:** In order to help the model distinguish stylistic nuances between preferred and non-preferred sonnets, we attempt a second round of fine-tuning using DPO. The DPO training objective uses a binary classification loss over preference pairs:

$$\mathcal{L}_{\text{DPO}}(\theta) = -\mathbb{E}_{(x, y_w, y_l)} \left[\log \sigma \left(\beta \left(\log \frac{\pi_\theta(y_w|x)}{\pi_{\text{ref}}(y_w|x)} - \log \frac{\pi_\theta(y_l|x)}{\pi_{\text{ref}}(y_l|x)} \right) \right) \right].$$

We first use a fine-tuned model to generate and score DPO data, and then train the model on the scored pairs. We experiment with different reward policies, such as rewarding purely based on high CHRF, or rewarding a coherent rhyme scheme and roughly correct token length/number of lines (e.g., we can score the examples using $1.5 * \text{word_count_score}(\text{text}) + 0.5 * \text{line_count_score}(\text{text}) + 1.0 * \text{line_length_score}(\text{text}) + 1.9 * \text{rhyme_scheme_score}(\text{text}) + 2 * \text{repetition_penalty}(\text{text})$). For scoring the rhyme scheme, we strip the last word from each line and use the pronouncing library [10] to check whether they obey the ABAB CDCD EFEF GG pattern.

Political Affiliation Detection: We utilize the same classifier as sentiment analysis, but apply it to a different classification task and dataset—namely, the `cajcodes/political-bias` dataset [11]. Since the data is not pre-split, we randomly divide the data according to a 70/15/15 ratio into train, dev and test sets respectively.

3.2 Optimizers

Adam: We complete the implementation of this optimizer, which maintains exponentially decaying estimates of both the first and second moments of the gradients. The update rule is

$$\theta_{t+1} = \theta_t - \alpha \frac{\hat{m}_t}{\sqrt{\hat{v}_t + \epsilon}}$$

where $\hat{m}_t$ and $\hat{v}_t$ are the bias-corrected first and second moment estimates of the gradients respectively.

Shampoo: We also implement the Shampoo optimizer, which generalizes diagonal second-moment methods like Adam by maintaining preconditioners for each layer’s weight matrix. Let G_t denote the gradient at step t for a weight matrix $W \in \mathbb{R}^{m \times n}$. Shampoo computes:

$$L_t = \sum_{i=1}^t G_i G_i^\top, \quad R_t = \sum_{i=1}^t G_i^\top G_i,$$

and updates the parameters via

$$W_{t+1} = W_t - \eta L_t^{-1/4} G_t R_t^{-1/4},$$

where $L_t^{-1/4}$ and $R_t^{-1/4}$ are matrix roots used to precondition the gradient, and η is the learning rate.

For the sentiment, paraphrase and sonnet tasks, we use the baselines given in Gradescope or in the DFP handout for measuring the efficacy of these approaches; for the custom political affiliation task, see 4.2 for baseline details.

3.3 Attention

FlashAttention: We update the original attention class to use the `torch.nn.functional.scaled_dot_product_attention()` function, which calls an implementation of FlashAttention in the backend, subject to system setup and constraints.

LoRA PEFT: We implement a new attention class that uses LoRA [12] to attempt to reduce RAM usage during training as well as training duration. Our implementation relies on freezing pretrained weights and splitting the matrix representing the actual updates as a product of two individual matrices, as shown in the diagram below. This implementation uses the overall equation

$$h = Wx + BAx.$$

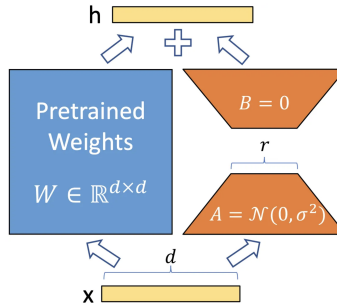


Figure 1: LoRA method. W is the original pretrained weight matrix, which is frozen. d is the full hidden size. A and B are low rank matrices.

4 Experiments and Results

4.1 Data

We use the following datasets:

- SST (8,544 train examples, 1,101 dev examples) for sentiment analysis [13].
- CFIMDB dataset (1,701 train examples, 245 dev examples) for sentiment analysis.
- Quora dataset (141,506 train examples, 20,215 dev examples) for paraphrase detection.

- Shakespearean sonnets (143 train sonnets, 12 test sonnets) for sonnet generation.
- The cajcodes/political-bias dataset (460 train examples, 99 dev examples) for political affiliation detection [11]. Its inputs are political sentences (e.g., "Cutting welfare programs is necessary to encourage personal responsibility.") with associated numerical outputs (0-4) indicating their political association, where 0 is the most "conservative" and 4 is the most "liberal."

4.2 Evaluation Method

For sentiment analysis, paraphrase detection, and sonnet generation, we use the following provided baselines:

- Last Linear Layer for SST Dev Accuracy: 0.462
- Full Model for SST Dev Accuracy: 0.513
- Last Linear Layer for CFIMDB Dev Accuracy: 0.861
- Full Model for CFIMDB Dev Accuracy: 0.976
- Paraphrase Dev: 0.882
- Paraphrase Test: 0.856
- Sonnet Dev: 41.414
- Sonnet Test: 41.208

For our custom political affiliation detection task, we use a baseline dev accuracy of .889, which we determined by running the task with GPT-2 with no fine-tuning.

4.3 Results

We achieve test scores of **0.868** for paraphrase detection and CHRF **42.712** for sonnet generation. We achieve best dev accuracies of **0.984** for sentiment analysis on the CFIMDB dataset and **0.530** on the SST dataset, and a best dev accuracy of **0.949** for political affiliation detection.

4.3.1 Sentiment Analysis

attention mechanism	dataset	size	lr	dropout	dev accuracy
Baseline	SST	Small	1e-5	.4	.515
Baseline	CFIMDB	Small	1e-5	.4	.951
Baseline	SST	Small	1e-5	.45	.515
Baseline	CFIMDB	Small	1e-5	.45	.984
scaled_dot_product()	SST	Small	1e-5	.45	.507
scaled_dot_product()	CFIMDB	Small	1e-5	.45	.955
LoRA	SST	Small	1e-5	.45	.502
LoRA	CFIMDB	Small	1e-5	.45	.959
LoRA	SST	Medium	1e-5	.45	.530
LoRA	CFIMDB	Medium	1e-5	.45	.967

Table 1: Sentiment analysis ablation studies. The model configurations yielding the best dev performances on the CFIMDB dataset and SST dataset are separately boxed. These best-performing model parameters were also used to generate the test leaderboard submissions.

We experiment with two pooling strategies for sentence-level representations: using the hidden state of the last token vs. mean pooling over all tokens with padding masked out. With the first strategy, the performance was consistently below baseline. After switching to the second strategy, we began to see more variation in performance, and were able to achieve dev accuracies higher than the baseline. All configurations in the table above use the mean pooling implementation.

4.3.2 Paraphrase Detection

size	lr	batch size	epochs	dev accuracy
small	1e-5	16	10	.897

Table 2: Best paraphrase detection results.

We were able to achieve a dev accuracy of .895 with our baseline implementation and all default finetuning parameters. After changing some of the finetuning parameters, we were able to achieve a slightly higher accuracy of .897. Other finetuning configurations resulted in noticeably worse accuracy.

4.3.3 Sonnet Generation

size	lr	patience	val	unfrozen	top k	temp	beam	ngrams	dataset	dev CHRF
small	1e-3	5	CHRF	3	50	1.2	no	2	extended	42.572
small	1e-3	5	CE	3	50	1.2	no	2	extended	41.675
small	1e-3	5	CHRF	2	50	1.2	no	2	extended	42.201
small	1e-3	5	CHRF	3	50	1.2	no	1	extended	42.276
small	1e-3	5	CHRF	3	50	1.2	yes	2	extended	40.946
small	1e-3	5	CHRF	3	50	1.2	no	2	original	41.597
small	1e-3	5	CHRF	3	30	1.2	no	2	extended	42.132
small	1e-3	5	CHRF	3	50	0.8	no	2	extended	41.576
small	1e-3	5	CHRF	3	50	1.75	no	2	extended	42.316
small	1e-3	5	CHRF	3	50	2	no	2	extended	42.510
small	1e-3	5	CHRF	3	50	2.5	no	2	extended	42.297
medium	1e-3	5	CHRF	3	50	1.2	no	2	extended	42.744
large	1e-3	5	CHRF	3	50	1.2	no	2	extended	42.707

Table 3: Sonnet ablation studies changing various parameters from the "default," shown in the topmost row, which is the best-performing GPT-2 small model configuration. Changed configurations and best dev performance are boxed. The best-performing dev model was also used to generate the test leaderboard submission.

Validation: We discover that validating on CHRF is, unsurprisingly, a more reliable predictor of final CHRF performance than traditional validation on cross-entropy loss (row 2 in the table 3).

Learning Rate: We discover that small initial learning rates, such as $1e-4$ or $1e-5$, exhibit slow convergence and sometimes converge to suboptimal minima. A learning rate of $1e-3$ is observed to make markedly better initial progress; although CHRF usually begins to decrease as the model experiences overfitting after about 4-5 epochs, we discover that our adaptive learning rate approach can restrain the model and force it to more thoroughly explore the area of potential convergence, often achieving a highest validation CHRF after the 10th epoch (!). (We also find that cosine annealing, surprisingly, underperforms the naive strategy.) Overall, with the adaptive learning rate strategy and CHRF validation, training typically took about 1 minute per epoch and 10-15 minutes per run.

Freezing: We discover that unfreezing about 3 layers results in the most optimal performance. When unfreezing 2 layers, the sonnets tend to be too prose-like, and when unfreezing significantly more layers, the sonnets become nonsensical.

Repetition Penalty: Naively applying the penalty (row 4) for the repetition of individual tokens significantly decreases the model's performance. However, as expected, penalizing the repetition of n-grams (we set it to 2 for most runs) improves the model's performance.

Beam Search: Using beam search in generation surprisingly decreases the performance of the model (row 5), sometimes qualitatively leading to more rambling or generic results.

Top-k: A top-k filter of 50 seems to be optimal. We observe that overstrict filtering often leads to prose-like rambling loops (row 7), while no top-k filtering leads to the inclusion of nonsense characters and modern words (e.g., "Medicare," "Costco").

Creating Data: The original dataset is quite small (see row 6 for original performance), so we use GPT-5.4 to generate more sonnets and extend it to 1000 items. Unsurprisingly, we discover that training the model on a larger dataset improves its performance.

Softmax Temperature: The softmax temperatures 1.2 and 2.0 achieve the best CHRF (see rows 8 to 11).

Model Size: Larger models in general tend to achieve higher scores than the default small model, although we did not observe a significant difference between the medium and large model presets. We also experiment with training the large model with LoRA and FlashAttention to save training time, and we observe that such runs perform about the same or slightly worse in terms of CHRF in comparison to the medium-sized model, as expected. See the last two rows above.

DPO: DPO does not seem to have a significant positive effect on the model. Generating DPO pairs using the model configuration in the first row of the table by evaluating 6 samples against our rhyme/length coherence policy (with the top response being "preferred" and the bottom response being "rejected") for each poem in the dataset seems to slightly improve the readability and genre adherence of sonnets, but decreases their CHRF score (which is somewhat expected). Generating DPO pairs based on a purely CHRF ranking

and then training DPO with $\beta = 0.01$ on top of the model for 1 epoch results in a CHRF of 42.157, which is still above the baseline but not an improvement. Increasing β or the number of epochs dramatically decreases the performance of the model.

4.3.4 Political Affiliation Detection

size	optimizer	lr	dropout	batch size	epochs	dev accuracy
small	Adam	1e-5	0.3	10	13	0.939
small	Shampoo	1e-5	0.3	10	13	0.949

Table 4: Best political affiliation detection results.

We experiment with two optimizers (Adam and Shampoo) and vary several hyperparameters including learning rate, dropout, batch size, and number of epochs. We search over learning rates $\{10^{-4}, 10^{-5}\}$, dropout values $\{0.2, 0.3, 0.4\}$, batch sizes $\{8, 10\}$, and training epochs $\{10, 13, 16\}$.

We note that a smaller learning rate, such as 1e-5, likely caused the model to perform better since political affiliation detection is a task in which nuances can be overlooked when using a larger learning rate. We also used 13 epochs instead of the default 10 achieve slightly higher accuracy. 16 epochs caused the accuracy to decrease, likely due to overfitting.

5 Analysis

5.1 Tasks

In addition to our quantitative results, we made the following qualitative task-specific observations about potential reasons behind the performance of our fine-tuning strategy for some of the tasks.

5.1.1 Sentiment Analysis

We had trouble both reaching and exceeding the baseline accuracy for the sentiment analysis task through finetuning parameters. We wanted to run experiments using the larger GPT-2 models, but were concerned about the amount of RAM and time they would take up. Thus, we decided to try alternate attention implementations to attempt to reduce RAM usage. Beyond the baseline attention implementation, we also tried an alternate implementation using the `torch.nn.functional.scaled_dot_product_attention()` function, and the LoRA PEFT implementation, which also uses the `torch.nn.functional.scaled_dot_product_attention()` function. Finally, we also experiment using the medium and large pretrained models, using a similar code structure for processing this new argument as in the starter code for `sonnet_generation.py`.

After examining both the SST and CFIMDB datasets, we found that the SST dataset’s labels are on a scale from 1-5, which inherently makes it more difficult to predict the correct score, as opposed to predicting a binary label for the CFIMDB dataset. Additionally, the CFIMDB dataset consists of longer sentences than the SST dataset and even full paragraphs. This provides much fuller context to the data and makes it more likely that the model makes a sentiment classification that most people would agree with. It then makes sense why the model’s performance on the SST dataset is consistently much lower than its performance on the CFIMDB dataset.

5.1.2 Sonnet Generation

We propose some intuitive explanations for our experimental ablation results.

Freezing: As described in Section 3, the nonsensical results for more unfrozen layers are probably because the more fundamental layers describing basic grammar and syntax are disrupted.

Repetition Penalty: The naive repetition penalty likely decreases CHRF because it discourages the use of common words, such as "I" and "for," which are necessary for normal sentence structure and whose omission may lead to more nonsensical outputs.

Beam Search: Using beam search in generation may decrease CHRF because beam search favors high-probability sequences that are less Shakespearean and more similar to GPT-2’s prior distribution of modern English. Indeed, the literature seems to agree that beam search is better applicable to use cases where there is one very high-probability expected output, like translation, than to creative tasks like sonnet generation.

Softmax Temperature: Interestingly, qualitative evaluation suggests that temperature 1.2 produces more actually intelligible sonnets than 2.0. Furthermore, although 0.8 (and 0.9, not shown) produce significantly lower CHRFs, they were qualitatively notably more likely to adhere to the sonnet rhyme and meter requirements. More on this later.

DPO: We suspect that, when β is too large or we train for too many epochs, the model over-optimizes for the exact DPO examples. Addressing the overall finding that DPO makes no difference, it is possible that the dataset is still too small for DPO to have a strong enough signal; it is also possible that the "rejected" responses (which are often rejected because they are too short, which is severely punished by CHRF) are so terrible that they dramatically push the model in the opposite direction, leading to some slightly confused and over-correcting results.

In general, we find that, of the many constraints of the sonnet genre, it is easiest to train the model to use Shakespearean vocabulary (almost any training strategy picked this up immediately) and write roughly 14 lines of poetry. It is a little harder to train the model to write thematically consistent and "artistic" content and hardest to train the model to follow the rhyme scheme and iambic pentameter. We attempted to use qualitative observations about the rhyme, meter and thematic continuity of sample sonnets generated during the training process to determine how the learning rate should be dynamically adjusted and how many epochs the model should train for. However, we found this strategy to be unreliable because the CHRF score of a sonnet does not necessarily correlate with its thematic consistency or adherence to sonnet rhyme and meter. For example, at the second softmax temperature optimum at ≈ 2 , the poems are qualitatively very incoherent, even though they score quite well in dev CHRF. (The only genre requirements that appear to be heavily weighted by CHRF are the Shakespearean vocabulary and length of the sonnet being about 110-120 tokens; this prioritization makes sense given the CHRF formula [9].) In the interest of assignment scoring, we focused on achieving a good CHRF—e.g., by using CHRF validation and CHRF-based DPO reward policy—but future work in this area should consider methods to specifically target rhyme and meter, such as our initial mixed DPO reward policy, and find a better method for evaluating thematic consistency. Future work should also consider confirming results by averaging CHRF across runs with different seeds, as we found that varying the seed during runs on the same dev dataset could have significant effects on CHRF (up to ± 1).

5.1.3 Political Affiliation Detection

Although running the political affiliation task using the Shampoo optimizer increased the dev accuracy, we were not able to run Shampoo on any other task due to its large RAM usage. We attempted to run Shampoo with LoRA as well, but the RAM usage was still too high for our VM setup. Thus, we ran a majority of tasks using the default Adam optimizer.

The Shampoo optimizer was likely able to improve the accuracy on the political affiliation task because the dataset we used had less than 500 training samples, which is far smaller than the dataset for any other task. Shampoo can adapt weights/parameters together, as opposed to the individual updates Adam performs, which could have allowed it to see subtle correlations between samples in the small dataset that Adam was not able to.

6 Conclusion

We found that finetuning parameters on the basic implementation for each task had the greatest improvements from the baseline. Our implementations of the Shampoo optimizer, LoRA PEFT, and DPO all slightly decrease the performance, given that we use the finetuning parameters that provided the best performance without the extensions. Our best performance is .897 on the paraphrase task, which is .015 higher than the baseline, .515 and .984 for the sentiment classification task on the SST and CFIMDB datasets, which is .002 and .008 higher than the baselines, respectively, and 42.744 dev CHRF and 42.712 test CHRF for sonnet generation, which are 1.33 and 1.504 higher than the baseline CHRFs, respectively.

In the future, one possible avenue of exploration would include using the GPT-2 Large model instead of GPT-2 for all tasks, which would require significantly more resources. Another possible avenue would be using models that have been finetuned on a specific task and testing their performance on a related, but different downstream task, as we saw in "Language Models are Unsupervised Multitask Learners" [1].

Team contributions

All members contributed to running experiments on various tasks using different finetuning parameters, writing the report, and creating the final poster.

Samantha Leventis did the initial completion of GPT-2 and the Adam optimizer, including the all the missing code blocks from `attention.py`, `gpt2_layer.py`, `gpt2.py`, and `optimizer.py`. She also implemented the mean-pooling sequence representation, Shampoo optimizer, and full political affiliation detection extension.

Anna Wu implemented `sonnet_generation.py` and all related extensions, including DPO. She also ensured all GCP VMs and environments were working as expected, wrote the utility scripts `test_cuda.py` and `plot_utils.py` and helped troubleshoot the other parts of the project.

Iris Xu implemented `classifier.py` and `paraphrase_detection.py`, including iterations of improvements for experiments. She also implemented extensions in `attention.py` and `gpt2_layer.py` using the `torch.nn.functional.scaled_dot_product_attention()` function, as well as LoRA PEFT.

References

- [1] Alec Radford, Jeff Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners, 2019. Available at: https://cdn.openai.com/better-language-models/language_models_are_unsupervised_multitask_learners.pdf.
- [2] Peter J Liu, Mohammad Saleh, Etienne Pot, Ben Goodrich, Ryan Sepassi, Lukasz Kaiser, and Noam Shazeer. Generating wikipedia by summarizing long sequences. *arXiv preprint arXiv:1801.10198*, 2018.

- [3] Vineet Gupta, Tomer Koren, and Yoram Singer. Shampoo: Preconditioned stochastic tensor optimization, 2018.
- [4] Tri Dao, Daniel Y. Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. Flashattention: Fast and memory-efficient exact attention with io-awareness, 2022.
- [5] Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. Lora: Low-rank adaptation of large language models, 2021.
- [6] Rafael Rafailov, Archit Sharma, Eric Mitchell, Stefano Ermon, Christopher D. Manning, and Chelsea Finn. Direct preference optimization: Your language model is secretly a reward model, 2024.
- [7] Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Holger Keith, Alexander Rush, Jeff Pauls, Guillaume Gautheron, Luke Boulanger, Marcell Ostendorff, Sun Xu, Lingzhu Zhu, Patrick von Platen, Matthieu Ma, Shanya Jern, Julien Plu, Fanqi Xu, Teven Le Scao, Sylvain Gugger, Mariama Delecourt, Arnaud Timothee, Louis Moisan, and Mehdi Jondy. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 38–45, Online, October 2020. Association for Computational Linguistics.
- [8] PyTorch Contributors. torch.optim.lr_scheduler.cosineannealinglr. https://docs.pytorch.org/docs/stable/generated/torch.optim.lr_scheduler.CosineAnnealingLR.html, 2024. Accessed: 2026-03-15.
- [9] Maja Popović. chrF: character n-gram f-score for automatic mt evaluation. In *Proceedings of the Tenth Workshop on Statistical Machine Translation*, pages 392–395, Lisbon, Portugal, 2015. Association for Computational Linguistics.
- [10] Allison Parrish. pronouncing: A python interface to the cmu pronouncing dictionary. <https://pypi.org/project/pronouncing/>, 2018. Version 0.2.0.
- [11] cajcodes. cajcodes/political-bias dataset. <https://huggingface.co/datasets/cajcodes/political-bias>.
- [12] Samuel Lina Braz. Peft: Parameter-efficient fine-tuning methods for llms. 2025. Available at: <https://huggingface.co/blog/samuellimabraz/peft-methods>.
- [13] Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D. Manning, Andrew Y. Ng, and Christopher Potts. Recursive deep models for semantic compositionality over a sentiment treebank, 2013.