

Building GPT-2: Revisiting a Key Milestone of NLP

Stanford CS224N Default Project

Andy Wang

Department of Computer Science
Stanford University
wangandy@stanford.edu

Darren Chan

Department of Computer Science
Stanford University
dchan04@stanford.edu

Derek Yan

Department of Math
Stanford University
yanderek@stanford.edu

Abstract

Large pretrained language models achieve strong performance across many NLP tasks, but adapting them through full fine-tuning can be computationally expensive, unstable with limited data, and unable to capture task-specific structural constraints. In this project, we implement a GPT-2 style decoder-only transformer and evaluate it on Sentiment Analysis, Paraphrase Detection, and Sonnet Generation. We investigate three extensions to improve adaptation: Low-Rank Adaptation (LoRA) for parameter-efficient fine-tuning, Smoothness-inducing Adversarial Regularization and Bregman Proximal Point Optimization (SMART) for stable optimization, and structural positional embeddings for modeling poetic structure. Our results show that prediction-space regularization substantially improves paraphrase detection accuracy, while structural embeddings improve sonnet form metrics, highlighting the importance of combining efficient fine-tuning methods, robust optimization, and task-specific inductive biases.

TA Mentor: Ali Khan No external collaborators, no external mentor, no shared project

Late Days:

Team member	Late days remaining
Andy Wang	3
Darren Chan	3
Derek Yan	3
Total pooled	9
Team size	3
Late days applied for the report	3

1 Introduction

Large-scale pretrained language models have become the dominant paradigm in natural language processing (NLP). Transformer-based architectures such as GPT-2 [1] demonstrate that training on large corpora of unlabeled text enables models to learn general language representations that can be adapted to a wide variety of downstream tasks. In this project, we implement a GPT-2 style decoder-only transformer and evaluate it on several downstream tasks including Sentiment Analysis, Paraphrase Detection, and Sonnet Generation. These tasks highlight both the generative capabilities of large language models and their flexibility in adapting to diverse NLP problems through task-specific fine-tuning.

Despite their success, adapting large pretrained models presents several practical challenges. First, modern language models contain hundreds of millions or billions of parameters, making full fine-tuning computationally expensive and memory intensive [2]. Second, fine-tuning can be unstable, especially when training data is limited,

potentially leading to overfitting or degraded generalization performance [3]. Finally, standard transformer architectures rely on positional encodings that capture token order but may not fully represent higher-level structural information present in language tasks such as poetry generation [4].

To address these challenges, we explore several extensions designed to improve the efficiency and robustness of fine-tuning in transformer-based language models. Specifically, we investigate three complementary approaches. First, we implement *Low-Rank Adaptation* (LoRA) [5], a parameter-efficient fine-tuning method that introduces trainable low-rank matrices into selected linear layers while keeping the original pretrained weights frozen. This approach significantly reduces the number of trainable parameters while still allowing the model to adapt to downstream tasks.

Second, we explore *SMART* regularization (Smoothness-inducing adversarial regularization and Bregman proximal point optimization) [6], which improves the stability and robustness of generated predictions through principled regularized optimization. SMART incorporates a Bregman proximal point optimization framework together with adversarial smoothing to encourage the model to produce stable predictions under small perturbations. This approach aims to reduce overfitting and improve generalization during fine-tuning.

Third, to address the inherent limitations of Byte-Pair Encoding (BPE) and standard sequence modeling in capturing poetic form, we introduce *structural positional embeddings* for the Sonnet Generation task. Standard absolute positional embeddings process text as a flat 1D sequence, which struggles to represent the 2D structure (lines and syllables) required for a proper Shakespearean sonnet. Drawing inspiration from hierarchical positional embeddings used in document modeling [7] and 2D grid encodings used in vision models [8], we explicitly project the absolute 1D sequence into 2D hierarchical components—line index and syllable position—which provides the attention mechanism the inductive biases necessary to better model iambic pentameter.

In this project, we first implement GPT-2 following the given project framework and evaluate its baseline performance on several downstream tasks. We then incorporate the three extensions described above—LoRA, SMART regularization, and structural positional embeddings—to investigate whether they can improve model efficiency, robustness, or task performance. Our experiments compare different training configurations and analyze how these techniques affect both quantitative performance and qualitative model behavior.

Overall, this work investigates how parameter-efficient adaptation, regularized optimization, and structural inductive biases can improve the fine-tuning of large language models. Through our experiments, we provide empirical insight into the effectiveness of these methods and their potential role in improving transformer-based NLP systems.

2 Related Work

Transformer-based language models. Recent advances in natural language processing have been driven by large transformer-based language models. The transformer architecture introduced by Vaswani et al. [4] relies on self-attention mechanisms to model long-range dependencies in sequences and has become the foundation for most modern NLP systems. Building on this architecture, autoregressive language models such as GPT [9] and GPT-2 [1] demonstrated that large-scale unsupervised pretraining on massive text corpora can produce models that generalize effectively to many downstream tasks after fine-tuning. These models learn contextual representations of language that can be adapted for classification, generation, and reasoning tasks.

Parameter-efficient Fine-Tuning. Several methods have been proposed to reduce the number of parameters that must be updated during fine-tuning. Adapter layers [10] introduce small bottleneck modules within each transformer block that are trained while the original model parameters remain frozen. Prefix tuning [11] instead learns continuous prompt vectors that are prepended to the input sequence and influence the attention mechanism. BitFit [12] shows that updating only the bias parameters of a pretrained model can achieve competitive performance on certain tasks. These approaches significantly reduce the number of trainable parameters while maintaining strong downstream performance.

Low-Rank Adaptation (LoRA). Low-Rank Adaptation (LoRA) [5] is a parameter-efficient fine-tuning technique that introduces trainable low-rank matrices into existing linear layers of a pretrained model. Instead of updating the full weight matrix, LoRA learns a low-rank update that is added to the frozen pretrained weights. This method greatly reduces the number of trainable parameters while preserving the expressiveness of the model. Subsequent work has further explored parameter-efficient adaptation strategies and extensions to LoRA, including improvements to low-rank adaptation for large language models and methods that combine low-rank updates with other efficient tuning strategies [13, 14]. These approaches demonstrate that large pretrained models can be effectively adapted to downstream tasks with minimal parameter updates, motivating the use of LoRA in our experiments.

Prediction-space Regularization and Optimization. Another line of work focuses on improving the robustness and stability of fine-tuning. SMART (Smoothness-inducing adversarial regularization and Bregman

proximal point optimization) [6] introduces a principled regularization framework based on Bregman proximal point optimization combined with Adversarial Smoothing. Bregman proximal point optimization stabilizes training by constraining each parameter update to remain close to the predictions of a frozen reference model from a previous training iteration. SMART builds on this to add smoothness-inducing adversarial regularization, an approach that encourages the model to produce stable predictions under small perturbations of the input or hidden representations, improving generalization and preventing overfitting during fine-tuning. This idea is closely related to broader research on adversarial training and stability-based regularization for neural networks [15, 16]. Motivated by these findings, we incorporate SMART regularization into the fine-tuning process to encourage stable and robust optimization.

Hierarchical Positional Encoding. It is well established that standard positional embeddings fail to capture the complex structure of long documents or the 2D dependence in vision transformer models (ViTs). Hierarchical positional encoding techniques have arisen to address this issue in documents [7], in vision [8], in tabular data [17], and more. This technique can also be extended to Shakespearean sonnets, where structural adherence is paramount, and we explore this direction in this work.

Together, these lines of work highlight three key directions for improving pretrained language models: efficient adaptation through parameter-efficient fine-tuning, robust optimization techniques for stable training, and the incorporation of structural inductive biases. Our work explores these directions by integrating LoRA, SMART regularization, and structural positional embeddings into a GPT-2 based framework.

3 Approach

This project builds on the provided GPT-2 skeleton code and completes the implementation of GPT-2 [1]. We first implement the core components of the GPT-2 architecture, including masked multi-head self-attention, GPT-2 transformer layers, embedding layers, and the AdamW optimizer [18]. We then fine-tune our GPT-2 implementation through a classification exercise, predicting the sentiment of various sentences on two datasets. Finally, we fine-tune GPT-2 for two downstream tasks: (1) cloze-style Paraphrase Detection on the Quora dataset and (2) Sonnet Generation conditioned on the first three lines, using Shakespeare’s sonnets as a dataset. The provided baseline is the small pretrained GPT-2 model (12 layers, 768 hidden dimension, 12 attention heads), fine-tuned using vanilla cross-entropy loss.

Sentiment Analysis. For Sentiment Analysis, we use the final hidden representation of the last token as a sentence-level embedding. Given an input sentence tokenized as $x_1, \dots, x_T$, GPT-2 produces contextualized hidden states

$$h_1, \dots, h_T = \text{GPT2}(x_1, \dots, x_T).$$

We use the final token representation h_T as a fixed-length sentence embedding, which is passed through a dropout layer and a linear classification head:

$$z = Wh_T + b,$$

where $W \in \mathbb{R}^{C \times d}$, $d = 768$, and C is the number of sentiment classes (5 for SST and 2 for CFIMDB). The predicted class distribution is then

$$P(y | x) = \text{softmax}(z).$$

Paraphrase Detection. For Paraphrase Detection, we follow a cloze-style formulation. Given a sentence pair (s_1, s_2) , we construct the prompt

```
Question 1: “s1”
Question 2: “s2”
Are these questions asking the same thing?
```

The model predicts either the token “yes” or “no” as the next token. The predicted label is therefore

$$\hat{y} = \arg \max_{y \in \{\text{yes}, \text{no}\}} P(y | s_1, s_2).$$

We interpret the output probability over the tokens corresponding to “yes” and “no” as

$$P(y | s_1, s_2) = \text{softmax}(Wh_T),$$

where h_T is the final hidden state of the last token and W projects the hidden representation to the vocabulary distribution.

Sonnet Generation. For Sonnet Generation, we train GPT-2 using the standard autoregressive language modeling objective. Given a sequence of tokens $x_1, \dots, x_T$, the model maximizes the log-likelihood

$$\sum_{t=1}^T \log P(x_t | x_1, \dots, x_{t-1}),$$

which corresponds to minimizing the cross-entropy loss between predicted and ground-truth tokens. During inference, we condition on the first three lines of a sonnet and generate subsequent tokens sequentially. To improve generation diversity and prevent degenerate outputs, we use temperature scaling together with top- k sampling.

Low-Rank Adaptation (LoRA). To improve parameter efficiency during fine-tuning, we implement Low-Rank Adaptation (LoRA) [5]. Instead of updating the full weight matrices of the pretrained GPT-2 model, LoRA introduces trainable low-rank matrices into selected linear layers while keeping the original pretrained weights frozen. Specifically, for a weight matrix $W \in \mathbb{R}^{d \times k}$, LoRA parameterizes the weight update as

$$W' = W + \frac{\alpha}{r} BA,$$

where $A \in \mathbb{R}^{r \times k}$ and $B \in \mathbb{R}^{d \times r}$ are trainable matrices with rank r , and α is a scaling factor. In our implementation, we apply LoRA to the query and value projection matrices of the self-attention mechanism. During training, only the LoRA parameters are updated while the pretrained backbone remains frozen. This greatly reduces the number of trainable parameters while still allowing the model to adapt its attention behavior for downstream tasks.

SMART Regularization (Adversarial Smoothing and Bregman Proximal Point Optimization).

In addition to parameter-efficient adaptation, we investigate SMART regularization [6] to improve the robustness and stability of fine-tuning. SMART formulates fine-tuning as a regularized optimization problem that encourages smooth model predictions under small perturbations of the input or hidden representations. The method builds on a Bregman proximal point optimization framework, which constrains parameter updates at each iteration to remain close to the previous model parameters. Specifically, at iteration $t + 1$, the vanilla Bregman proximal point (VBPP) update takes the form

$$\theta_{t+1} = \arg \min_{\theta} F(\theta) + \mu D_{\text{Breg}}(\theta, \theta_t),$$

where $F(\theta)$ is the task-specific loss, $\mu > 0$ is a regularization coefficient, and $D_{\text{Breg}}(\cdot, \cdot)$ denotes the Bregman divergence. In the SMART framework, the Bregman divergence is defined over the model outputs as

$$D_{\text{Breg}}(\theta, \theta_t) = l_s(f(\tilde{x}_i; \theta), f(x_i; \theta_t)),$$

where $f(\cdot; \theta)$ denotes the model prediction function and $\tilde{x}_i$ represents a perturbed version of the input x_i . This regularization encourages the model to produce consistent predictions for both the original and perturbed inputs, thereby improving robustness and preventing overly aggressive parameter updates during fine-tuning. In practice, this results in smoother decision boundaries and improved generalization performance when adapting large pretrained language models to downstream tasks.

To enable independent control and allow for ablation testing, our implementation modifies this original formulation by decoupling Adversarial Smoothing and the Bregman update into two additive loss components. Furthermore, solving the inner maximization problem posed by SMART iteratively (e.g., via multi-step Projected Gradient Descent as in the original formulation) incurs severe computational overhead. To avoid this, we approximate the worst-case perturbation using a single-step gradient ascent update [15, 16]. This efficiently generates the adversarial penalty as an additive loss term without the need to solve a costly iterative optimization problem at every training step.

Structural Positional Embeddings. Finally, we experiment with structural positional embeddings to incorporate task-specific structural information into the transformer architecture. Standard positional embeddings encode the sequential position of tokens but do not explicitly represent higher-level structural features. For the Sonnet Generation task, poetic form includes structural constraints such as line positions and syllable patterns. To capture this information, the final input embedding E_{total} for a given token is computed as:

$$E_{\text{total}} = E_{\text{token}} + E_{\text{absolute_position}} + \alpha_{\text{line}} E_{\text{line}} + \alpha_{\text{syllable}} E_{\text{syllable_position}}$$

where E_{token} and $E_{\text{absolute_pos}}$ denote the standard BPE token and learned absolute positional embeddings, respectively. The terms $E_{\text{line_num}}$ and $E_{\text{syllable_pos}}$ represent distinct, learned embedding layers that encode the token’s current line index and its syllable position within that line. Furthermore, we introduce the learnable scalar weights α_{line} and α_{syllable} to modulate the influence of these structural signals. By explicitly exposing this higher-level architectural structure to the attention mechanism, the model may be better equipped to natively satisfy the rigid phonetic and formatting constraints of a Shakespearean sonnet. Syllable counts are calculated using a combination of canonical pronunciations in CMUdict [19] and a rule-based heuristic [20]. Rhymes are calculated using phoneme-based comparisons of canonical pronunciations in CMUdict [19], matching the final stressed vowel and subsequent phonemes.

4 Experiments

4.1 Data

For Sentiment Analysis, we evaluate on two benchmark datasets. First, the Stanford Sentiment Treebank (SST) [21] consists of 11,855 movie review sentences annotated with five sentiment labels: negative, somewhat negative, neutral, somewhat positive, and positive. We use the standard train/dev/test splits (8,544 / 1,101 / 2,210 examples). The task is 5-way sentence-level classification. In addition, the CFIMDB dataset contains 2,434 highly polar movie reviews labeled as either positive or negative. The dataset is split into train/dev/test (1,701 / 245 / 488 examples). Unlike SST, reviews are often multi-sentence and substantially longer.

For Paraphrase Detection, we use the Quora Question Pairs dataset [22]. The dataset contains question pairs labeled as paraphrase (1) or non-paraphrase (0). We use the provided subset with 141,506 training examples, 20,215 development examples, and 40,431 test examples. We formulate this as a cloze-style generation task, where the model predicts the token “yes” or “no” given a sentence pair prompt.

For Sonnet Generation, we will use the Shakespeare sonnet dataset consisting of 154 sonnets, each containing 14 lines following iambic pentameter and the ABAB CDCD EFEF GG rhyme scheme. The data is split into 143 training sonnets and 12 test sonnets. During evaluation, the first three lines of each test sonnet are provided as context, and the model generates the remaining lines autoregressively.

4.2 Evaluation Method

For both Sentiment Analysis and Paraphrase Detection, we evaluate classification using accuracy, which is the proportion of correct predictions out of all predictions.

For Sonnet Generation, we evaluate using the chrF metric [23], a character-level n -gram F-score that measures similarity between generated text and reference sonnets. Let P and R denote character n -gram precision and recall. Then, chrF computes the following, where β controls the precision-recall tradeoff:

$$\text{chrF} = \frac{(1 + \beta^2)PR}{\beta^2 P + R}. \quad (1)$$

We also evaluate the level of compliance of the generated sonnets with Shakespearean form by calculating the percent of generated sonnets with exactly 14 lines, the fraction of lines with exactly 10 syllables, the average number of syllables per line, and the percent of required rhymes that were correct.

4.3 Experimental Details

For Sentiment Analysis on the SST dataset training only the last linear layer, we used a batch size of 64, learning rate of $3e-3$, and decreased hidden dropout probability to 0.1 since we were only training a small linear layer. For fine-tuning the full model, we used a batch size of 64 and a learning rate of $3e-5$. For the CFIMDB dataset, for fine-tuning only the last linear layer, we used a batch size of 8, learning rate of $3e-3$, and hidden dropout probability of 0.3. For fine-tuning the full model on CFIMDB, we use a batch size of 8, learning rate of $3e-5$, and hidden dropout probability of 0.3. For both datasets, we trained the last linear layer for 10 epochs and trained the full model for 5 epochs.

For Paraphrase Detection, we did an initial grid search over learning rate, batch size, and SMART, Bregman, and LoRA parameters on low epoch numbers to find the the best hyperparameters to train for longer epochs. After this hyperparameter grid search, we selected the following hyperparameters: batch size of 8, learning rate of 10^{-5} , and trained for 8 epochs. When Adversarial Smoothness was employed, we used a perturbation radius ϵ of 1×10^{-5} , an adversarial step size of 1×10^{-3} and a loss weight λ_S of 0.1. When Bregman was used, we used μ of 0.01. Due to the fact that Adversarial Smoothing and Bregman introduce additional forward and backward passes that substantially increase training time, when LoRA was used, we continued using the GPT2 small model rather than larger versions in order to allow for training more epochs.

For Sonnet Generation, we performed a hyperparameter grid search over learning rate, batch size, various sampling strategies (top- p , top- k , beam search), structural embedding parameters, and LoRA parameters. We selected the following hyperparameters: a batch size of 2, learning rate of 1.5×10^{-5} , 10 epochs, top- p sampling with $p = 0.85$, and early stopping with patience of 2. When LoRA was employed, we switched to the GPT2-large model and used a learning rate of 2×10^{-5} , LoRA rank of 16, alpha of 32, dropout of 0.05, and applied the low-rank updates exclusively to the query and value projection matrices within the attention layers. The other hyperparameters remained the same. When structural positional embeddings were added, we used a learning rate multiple of 20 and alphas of 0.2 and 0.1 for GPT2 baseline and GPT2-Large + LoRA, respectively.

Method	Accuracy
GPT2 Baseline	0.862
GPT2 + Bregman	0.897
GPT2 + LoRA	0.868
GPT2 + Adversarial Smoothing + Bregman (full SMART)	0.893
GPT2 + Adversarial Smoothing + Bregman (full SMART) + LoRA	0.830

Table 1: Ablation testing on Paraphrase Detection measured by accuracy.

4.4 Results

Paraphrase Detection. Our best performing model was GPT2 with Bregman proximal regularization, which we submitted to the test set evaluation and achieved an accuracy of 0.896. The full SMART formulation achieved the second highest performance by a narrow margin with an accuracy of 0.893. Though we expected the full SMART formulation to perform better Bregman proximal regularization only, both of these results are improved significantly over the baseline, as expected. What was most unexpected was the underperformance of LoRA—though LoRA on its own slightly improved the baseline accuracy, SMART and LoRA together resulted in the worst performance, which was not expected. This indicates that for the task of Paraphrase Detection, regularization in the prediction space meaningfully improves results, but constraining the number of trainable parameters leads to performance decreases (see Analysis section5 for more details).

Method	chrF	% w/ 14 lines	% Lines w/ 10-syl.	Avg. # Syllables	% Rhymes Correct
GPT2 Baseline	42.10	0.25	24.38	10.69	5.95
GPT2 + Structural Embeddings	41.79	0.33 [†]	27.16 [†]	10.15 [†]	8.33 [†]
GPT2-Large + LoRA	41.85	0.50	35.15	10.30	13.10
GPT2-Large + LoRA + Structural Embeddings	40.88	0.25	37.65[†]	9.90[†]	13.10

[†] indicates sonnet form metrics that improve with structural embeddings.

Table 2: Ablation testing on Sonnet Generation measured by chrF and sonnet form metrics on the dev set.

Sonnet Generation. Our best performing model was GPT2 baseline after hyperparameter tuning, when assessed by chrF, which we submitted to the test set evaluation and achieved a chrF score of 42.57. This exceeded the model’s performance on the dev set, illustrating that we stopped training before the model could overfit to the training data. Due to the limit on submissions to the test set, we are unable to provide a comparison of the performance of our various approaches on the test set. We notice that fine-tuning with LoRA resulted in lower chrF scores, which is expected due to LoRA acting as a regularizer. We also observe that structural embeddings achieved better performance on sonnet form metrics, which is also expected. We provide more details and explanation in the Analysis section5.

5 Analysis

Paraphrase Detection. During early training, we observed strong overfitting on our baseline model, indicated by training loss decreasing monotonically while dev accuracy plateaued and began to decline. Initial attempts at adjusting hyperparameters such as weight decay or decreasing batch size led to underfitting and did not improve performance. Only after we introduced our prediction-space regularization techniques were we able to see stronger performance on the development set. Our best model used Bregman Proximal Point Regularization, suggesting that constraining the model’s predictions to remain close to those of a reference model stabilizes the fine-tuning process and prevents overly aggressive parameter updates, thus reducing the overfitting we experienced earlier in our approach.

Even though it did not improve dev accuracy more than using Bregman alone, adding Adversarial Smoothing together with Bregman regularization (the full SMART configuration) still significantly improved performance over the baseline, indicating that combining these two regularization frameworks promotes smoother and more stable model behavior. While the complete SMART formulation showed to be effective, the slightly lower accuracy relative to Bregman alone may reflect the stronger regularization imposed by Adversarial Smoothing. By forcing the model to remain robust to worst-case perturbations of the input embeddings, Adversarial Smoothing introduces more aggressive regularization. This effect is likely amplified in our experimental setting due to computational constraints. SMART requires additional forward passes and gradient computations to construct adversarial perturbations, substantially increasing the training time per epoch by about 3-4 times in our testing. As a result, we were only able to feasibly train the SMART configurations for a limited number of epochs.

Under longer training schedules, the model would have more opportunity to adapt to the stronger regularization introduced by adversarial smoothing, potentially recovering the small accuracy gap and even surpassing the performance of Bregman regularization alone.

LoRA showed the least performance benefits—it had a very modest improvement in dev accuracy when used alone, yet in our ablation studies, it led to the worst performance of all tested methods when combined with SMART, falling well below the baseline. One possible explanation is that LoRA already restricts the space of allowable parameter updates, and the addition of strong prediction-space regularization further limits the model’s flexibility. The interaction between all of these constraints may therefore lead to underfitting, preventing the model from adequately adapting to the Paraphrase Detection task. It is worth noting, however, that given a resource-limited environment, these effects are most pronounced. As an example, even though LoRA restricts the number of trainable parameters, for the task of Paraphrase Detection, using LoRA on GPT2-small was still the only tractable option within limited compute resources. This likely contributed significantly to LoRA’s underwhelming performance, as it is already operating in a restricted parameter space compared to GPT2-medium or GPT2-large. With more compute resources, it is likely that LoRA applied to larger base models does in fact provide sufficient representational capacity to satisfy the regularization constraints without suffering underfitting.

Sonnet Generation. First, we observed that substituting a larger base model and fine-tuning with LoRA achieved slightly lower chrF scores. This was somewhat anticipated, as LoRA is known to act as a regularizer and thus prevents the model from diverging too far from its pre-trained state. For Sonnet Generation, a model needs to output a highly specific and narrow distribution of text. Shakespeare only wrote a handful of sonnets after all. Full parameter fine-tuning of GPT2 likely allowed it to fit more tightly to the training distribution, while GPT2-Large + LoRA had greater inertia that resisted this, reflecting a tradeoff between expressiveness and parameter efficiency.

However, we did observe that GPT2-Large + LoRA possessed much better form adherence out of the box, with significant improvements over GPT2 in all four sonnet form metrics under investigation. This is likely due to GPT2-Large’s greater capacity for attentional understanding. Specifically, GPT2-Large has 20 attention heads and a 1,280 hidden dimension size compared to 12 heads and 768 hidden dimension size for GPT2. This enhances GPT2-Large’s ability to recognize syllable, line, rhyme, and other structural patterns in the training text. One head might look for phonemes that rhyme, another might keep count of syllables, and another might track the position of newline characters. If we extend this trend, we can expect that larger models perform even better at sonnet structure adherence.

Another result was that structural positional embeddings were effective at getting the models to generate more well-formed sonnets. The addition of the embedding layers improved all 4 form metrics for the GPT2 model and 2 of the 4 form metrics for the GPT2-Large model, with a third metric staying unchanged. This validates the hypothesis that well-implemented inductive hypotheses about the structure of domain-specific text is effective at steering generation towards a more desired state. However, there is a limit to how much steering one can inject into the model, as we observed that past a certain point, further perturbations to the embeddings by the structural positional layers significantly impaired chrF performance.

Furthermore, during hyperparameter tuning, we observed high variability in both dev chrF scores and sonnet form metrics from just small perturbations to key hyperparameters. We also observed that there was not a monotonic relationship between dev loss and dev chrF score. Our initial assumption that the two metrics would correlate, as seen in many simpler machine learning tasks like accuracy in classification, did not hold for the Sonnet Generation task. This could be due to the token priors for next-token prediction being too modern and thus misaligned from the corpus of Shakespearean language that the chrF score is calculated from.

The primary evaluation metric employed for the Sonnet Generation task is chrF. While chrF serves as a useful baseline for evaluating vocabulary alignment, it is blind to the strict structural and phonological constraints of the Sonnet Generation task. Because chrF relies strictly on character n-gram overlap with a reference text, chrF rewards unstructured Shakespearean prose while penalizing perfectly metered sonnets that use non-Shakespearean vocabulary. To provide a more task-aligned evaluation, we introduced a suite of custom sonnet form metrics that capture the need for a sonnet to be exactly 14 lines of iambic pentameter with a strict rhyme scheme. This shifts the evaluation from superficial text-matching to structural integrity, an interpretable indicator of generative success that is better aligned with the complex structural objective of this task. Quantitatively, we observe a tradeoff between chrF and form adherence, with the addition of structural positional embeddings decreasing chrF and improving form metrics across the board in both architectural sizes. This illustrates that the extra embedding layers impede on the model’s ability to output Shakespearean sounding language while steering it towards more structured generation.

Qualitatively, we observed that the models had an easier time rhyming the couplet at the end of sonnets than the quatrain groups 1. Intuitively, couplets are easier to rhyme since the rhyming lines come one after the other, while in a quatrain, the rhyming scheme alternates every other line. Rhyming quatrains requires the model to attend over a greater distance than for a couplet.

⋮

Third
Quatrain

Into false alluring sicknesses,
 And think such growth in fluttering feathers could pay
 A contrary profit for honest rhyme
 To make it write down a soldiery mast.
 True love hath eyes and kills spies with lies, ✓
 And maids that would tell lies for a sun could die. ✓

Rhyming
Couplet

Figure 1: Excerpt of poem generated by GPT2-Large + LoRA + Structural Embeddings.

6 Conclusion

In this project, we implemented GPT-2 from the provided skeleton code and evaluated it on Sentiment Analysis, Paraphrase Detection, and Sonnet Generation, while also exploring three extensions: LoRA for parameter-efficient fine-tuning, SMART regularization for more stable optimization, and structural positional embeddings for stronger inductive bias in poem generation. Our results showed that these extensions had task-dependent benefits: for Paraphrase Detection, both vanilla Bregman proximal regularization and SMART substantially improved over the baseline, indicating that stability-focused regularization and adversarial smoothing are especially effective for discriminative tasks with limited data. For Sonnet Generation, the baseline GPT-2 model achieved the strongest chrF score, while GPT2-Large + LoRA + Structural Positional Embeddings improved several form-based metrics such as 14-line adherence, syllabic conformity, and rhyme correctness, revealing a tradeoff between surface-level similarity and structural faithfulness to real Shakespearean sonnets. Overall, we learned that different downstream tasks require different forms of inductive bias and that common optimization or evaluation assumptions do not always transfer cleanly across settings. Our main achievements were successfully implementing the full GPT-2 pipeline, integrating and evaluating three meaningful extensions, and conducting ablation studies that clarified when these methods improved performance. At the same time, our conclusions are limited by compute constraints, the small size of the sonnet dataset, and the shortcomings of chrF as a poetry metric. Future work could explore larger backbone models for LoRA, better tuning of SMART’s regularization terms, and decoding or training objectives that explicitly enforce rhyme and meter.

Team contributions

Darren Chan was in charge of implementing the SMART extension for the task of Paraphrase Detection. Andy Wang was in charge of implementing the structural embeddings extension for the task of Sonnet Generation. Derek Yan was in charge of implementing the LoRA extension for the tasks of Paraphrase Detection and Sonnet Generation. All team members contributed equally to all other aspects of the literature search, codebase, running experiments, and written report.

References

- [1] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. https://cdn.openai.com/better-language-models/language_models_are_unsupervised_multitask_learners.pdf, 2019. OpenAI Technical Report.
- [2] Zeyu Han, Chao Gao, Jinyang Liu, and Jeff Zhang. Parameter-efficient fine-tuning for large models: A comprehensive survey. *arXiv preprint arXiv:2403.14608*, 2024.
- [3] Marius Mosbach, Maksym Andriushchenko, and Dietrich Klakow. On the stability of fine-tuning bert: Misconceptions, explanations, and strong baselines. *arXiv preprint arXiv:2006.04884*, 2020.
- [4] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2017.
- [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. *arXiv preprint arXiv:2106.09685*, 2021.
- [6] Haoming Jiang, Pengcheng He, Weizhu Chen, Jianfeng Gao, and Tuo Zhao. Smart: Robust and efficient fine-tuning for pre-trained natural language models through principled regularized optimization. In *Annual Meeting of the Association for Computational Linguistics (ACL)*, 2020.

- [7] He Bai, Peng Shi, Jimmy Lin, Luchen Tan, Kun Xiong, Wen Gao, and Ming Li. Segabert: Pre-training of segment-aware BERT for language understanding. *CoRR*, abs/2004.14996, 2020.
- [8] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale. *CoRR*, abs/2010.11929, 2020.
- [9] Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. Improving language understanding by generative pre-training. *OpenAI Technical Report*, 2018.
- [10] Neil Houlsby, Andrei Giurgiu, Stanislaw Jastrzebski, Bruna Morrone, Quentin De Laroussilhe, Andrea Gesmundo, Mohammad Attariyan, and Sylvain Gelly. Parameter-efficient transfer learning for nlp. In *International Conference on Machine Learning (ICML)*, 2019.
- [11] Xiang Lisa Li and Percy Liang. Prefix-tuning: Optimizing continuous prompts for generation. In *Annual Meeting of the Association for Computational Linguistics (ACL)*, 2021.
- [12] Elad Ben Zaken, Yoav Goldberg, and Shauli Ravfogel. Bitfit: Simple parameter-efficient fine-tuning for transformer-based masked language models. *arXiv preprint arXiv:2106.10199*, 2022.
- [13] Junxian He, Chunting Zhou, Xuezhe Ma, Taylor Berg-Kirkpatrick, and Graham Neubig. Towards a unified view of parameter-efficient transfer learning. *International Conference on Learning Representations*, 2022.
- [14] Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. Qlora: Efficient finetuning of quantized llms. *Advances in Neural Information Processing Systems*, 2023.
- [15] Ian Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. *International Conference on Learning Representations*, 2015.
- [16] Takeru Miyato, Andrew Dai, and Ian Goodfellow. Adversarial training methods for semi-supervised text classification. *International Conference on Learning Representations*, 2017.
- [17] Jonathan Herzig, Pawel Krzysztof Nowak, Thomas Müller, Francesco Piccinno, and Julian Martin Eisen-schlos. TAPAS: weakly supervised table parsing via pre-training. *CoRR*, abs/2004.02349, 2020.
- [18] Ilya Loshchilov and Frank Hutter. Fixing weight decay regularization in adam. *CoRR*, abs/1711.05101, 2017.
- [19] Carnegie Mellon University Speech Group. CMU pronouncing dictionary. <https://github.com/cmuspinx/cmudict>, 2014.
- [20] Michael Holtzsch. Counting syllables in english words. <https://mholtzsch.github.io/2018/05/29/syllables/>, May 2018.
- [21] 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. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1631–1642, 2013.
- [22] Quora Data Team. First Quora Dataset Release: Question Pairs. <https://quoradata.quora.com/First-Quora-Dataset-Release-Question-Pairs>, 2017. Online; public release of question pairs for duplicate/semantic equivalence detection.
- [23] Maja Popović. chrF: Character n-gram f-score for automatic mt evaluation. In Ondřej Bojar, Rajan Chatterjee, Christian Federmann, Barry Haddow, Chris Hokamp, Matthias Huck, Varvara Logacheva, and Pavel Pecina, editors, *Proceedings of the Tenth Workshop on Statistical Machine Translation*, pages 392–395, Lisbon, Portugal, September 2015. Association for Computational Linguistics.