

Build GPT-2

Stanford CS224N Default Project

Yushi Feng

Department of Computer Science
Stanford University
yushif@stanford.edu

Yuchan Guo

Department of Computer Science
Stanford University
yuchan01@stanford.edu

Abstract

Large pretrained language models provide a unified modeling framework, yet their effectiveness across fundamentally different task types remains an open practical question. In this project, we study the transferability of a decoder-only GPT-2 architecture to both discriminative and generative NLP settings. We implement the core model and optimizer from scratch, initialize with pretrained weights, and adapt the same backbone to sentiment classification, paraphrase detection, and Shakespearean sonnet generation. Our experiments show that task-aligned adaptation strategies are critical: full-model fine-tuning improves performance on long-form binary sentiment data, while lightweight architectural changes yield strong gains for paraphrase detection. For generation, partial fine-tuning combined with decoding constraints improves stylistic coherence on a small poetry corpus. Overall, our results demonstrate that a single pretrained autoregressive model can support diverse NLP objectives when optimization and task formulation are carefully designed.

1 Key Information to include

- Mentor: Fang Wu
- External Collaborators (if you have any): NA
- Sharing project: NA
- Late Days (if you are using any):

Team member	Late days remaining
Yushi Feng	5
Yuchan Guo	5
Total pooled	6
Team size	2
Late days applied for the report	3

2 Introduction

Pretrained language models have reshaped modern NLP by enabling a single architecture to support a wide range of downstream tasks through fine-tuning. While encoder-based models such as BERT have shown strong performance on classification benchmarks, decoder-only models like GPT-2 are primarily optimized for autoregressive generation. This raises an important practical question: to what extent can a generative pretrained model be effectively adapted to tasks that differ substantially from its original training objective?

In this project, we investigate this question by applying GPT-2 to three distinct NLP settings: sentiment classification, paraphrase detection, and structured text generation. These tasks differ in their modeling requirements. Sentiment analysis demands reliable label prediction under varying supervision regimes,

paraphrase detection requires fine-grained semantic comparison between sentence pairs, and sonnet generation requires coherent long-form continuation with stylistic constraints. Studying all three within a unified framework allows us to analyze how architectural design, optimization strategy, and task formulation jointly influence transfer performance.

To enable systematic experimentation, we implement the core GPT-2 architecture and AdamW optimizer, and initialize our implementation using pretrained checkpoints. We then adapt the same backbone to each task using lightweight task-specific modifications, ranging from classifier heads for discriminative tasks to autoregressive fine-tuning for generation. Through this setup, we aim to better understand the practical trade-offs involved in transferring a decoder-only Transformer across heterogeneous NLP objectives.

3 Related Work

Our project is most directly connected to the line of work on generative pretraining introduced by GPT and expanded by GPT-2. These papers showed that a decoder-only Transformer trained with next-token prediction can transfer to many downstream NLP tasks, which motivates our use of a single pretrained GPT-2 backbone across sentiment analysis, paraphrase detection, and sonnet generation.^{1 2}

For discriminative tasks such as sentiment classification and semantic matching, encoder-based models like BERT are often a more natural fit. BERT and GLUE established sentence classification and sentence-pair understanding as standard transfer-learning evaluations, providing useful context for our sentiment and paraphrase experiments.^{3 4} Our project differs in that we apply a decoder-only model to these tasks, making task formulation and fine-tuning strategy especially important.

Our paraphrase method is also related to work that casts classification as cloze-style prediction. Prompt-based formulations such as PET show that pretrained language models can solve classification tasks by predicting label words in natural-language patterns.⁵ The provided baseline follows this idea, while our final model moves closer to standard supervised fine-tuning by replacing full-vocabulary prediction with a binary classification head. On the optimization side, our use of AdamW, warmup scheduling, and gradient clipping follows standard Transformer training practice.⁶

For sonnet generation, prior work has shown that poetry generation is more demanding than ordinary language modeling because it requires style, coherence, and often formal structure such as rhyme or meter.⁷ Compared with these specialized poetry systems, our approach is simpler: we evaluate how far a general-purpose GPT-2 model can go through fine-tuning alone.

4 Approach

We implement the core GPT-2 architecture and optimizer, initialize the model with official pretrained weights, and adapt the same backbone to sentiment classification, paraphrase detection, and sonnet generation. This section describes both the shared model foundation and the task-specific methods used in our experiments. Unless otherwise noted, the overall project scaffold, dataset interfaces, and evaluation scripts were provided by the course starter code, while the GPT-2 model components, optimizer, and task-specific modeling logic described below were implemented by our team.

4.1 Basic Model Development

The foundation of our system is a decoder-only GPT-2 Transformer⁸. We implemented the core model in our own codebase, including token embeddings, positional embeddings, causal multi-head self-attention, transformer blocks, final layer normalization, and autoregressive masking. We then loaded official pretrained GPT-2 weights using the HuggingFace checkpoint interface to initialize our implementation.

Given an input token sequence $x = (x_1, \dots, x_n)$, the embedding layer constructs the initial hidden states as

$$h_t^{(0)} = E[x_t] + P[t], \quad (1)$$

where E is the token embedding matrix and P is the learned positional embedding table. The hidden states are then passed through L stacked GPT-2 layers. In each layer, causal self-attention is computed

as

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^\top}{\sqrt{d_k}} + M\right)V, \quad (2)$$

where M is the causal mask that prevents each position from attending to future tokens.

Our transformer blocks follow the pre-layer-normalization structure used in GPT-2. For layer ℓ , the update is

$$\tilde{h}^{(\ell)} = h^{(\ell-1)} + \text{Dropout}\left(\text{MHA}(\text{LN}(h^{(\ell-1)}))\right), \quad (3)$$

$$h^{(\ell)} = \tilde{h}^{(\ell)} + \text{Dropout}\left(\text{MLP}(\text{LN}(\tilde{h}^{(\ell)}))\right). \quad (4)$$

After the final layer norm, we take the hidden state of the last non-padding token as the sequence representation:

$$r(x) = h_n^{(L)}. \quad (5)$$

This last-token representation is the shared feature vector used by our downstream classification tasks.

We also implemented the optimizer in `optimizer.py`. Our update rule follows AdamW. Given gradient g_t at step t , the first and second moments are updated as

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1)g_t, \quad v_t = \beta_2 v_{t-1} + (1 - \beta_2)g_t^2. \quad (6)$$

With bias correction,

$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t}, \quad \hat{v}_t = \frac{v_t}{1 - \beta_2^t}, \quad (7)$$

the parameter update becomes

$$\theta_t \leftarrow \theta_{t-1} - \alpha \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon} - \alpha \lambda \theta_{t-1}, \quad (8)$$

where α is the learning rate and λ is the decoupled weight decay coefficient. This optimizer implementation is shared across all downstream experiments.

4.2 Sentiment Analysis

For sentiment analysis, we adapt GPT-2 into a sentence classifier using the final token representation $r(x)$. We evaluate on two required datasets: SST, which is a five-class sentiment classification task, and CFIMDB, which is a binary sentiment classification task over longer reviews. Our sentiment classifier is implemented in `classifier.py`.

The model architecture is deliberately simple. After obtaining the last-token representation from GPT-2, we apply dropout and a linear classifier:

$$z_{\text{sent}} = W_{\text{sent}} \cdot \text{Dropout}(r(x)) + b_{\text{sent}}, \quad (9)$$

where $z_{\text{sent}} \in \mathbb{R}^C$ and $C = 5$ for SST or $C = 2$ for CFIMDB. The predicted class distribution is

$$p(y | x) = \text{softmax}(z_{\text{sent}}), \quad (10)$$

and training minimizes the cross-entropy loss

$$\mathcal{L}_{\text{sent}} = -\log p(y | x). \quad (11)$$

We follow the two fine-tuning settings required by the project. In the `last-linear-layer` setting, all GPT-2 parameters are frozen and only the task-specific classifier head is trained. In the `full-model` setting, the entire pretrained backbone is updated together with the classifier head. This setup allows us to test whether fixed pretrained representations are sufficient for sentiment analysis or whether end-to-end adaptation provides additional gains.

The sentiment model itself is close to the provided baseline in task formulation, but it depends on our own GPT-2 implementation and AdamW optimizer. We therefore view sentiment analysis as both a benchmark task and a validation that our core model implementation transfers properly to supervised classification.

4.3 Paraphrase Detection

The paraphrase task asks whether two questions express the same meaning. The provided baseline formulates this as a cloze-style prediction problem using an input prompt of the form

$$\text{Is “}s_1\text{” a paraphrase of “}s_2\text{”? Answer “yes” or “no”} \quad (12)$$

and then predicts the next token. In the baseline, the final hidden state is projected to the full GPT-2 vocabulary using weight tying:

$$z_{\text{base}} = r(x)E^\top \in \mathbb{R}^{|V|}, \quad (13)$$

where $|V|$ is the vocabulary size. The model is then trained so that the correct next token is either the GPT-2 token for yes or the token for no. This baseline is aligned with GPT-2’s language modeling interface, but it is inefficient for a binary decision problem because the model must choose between two relevant tokens and tens of thousands of irrelevant ones.

Our final paraphrase model keeps the same prompt-based input formulation but replaces the full-vocabulary prediction layer with a dedicated binary classification head. Specifically, we compute

$$z_{\text{para}} = W_{\text{para}} \cdot \text{Dropout}(r(x)) + b_{\text{para}}, \quad z_{\text{para}} \in \mathbb{R}^2, \quad (14)$$

where the two logits correspond to the classes no and yes. To remain compatible with the existing evaluation code, we place these two logits back into the vocabulary slots corresponding to the GPT-2 token ids for no and yes, while assigning all other vocabulary entries a very large negative value. This preserves the cloze-style evaluation interface but learns a much more direct binary decision boundary.

This change is the main modeling improvement over the starter baseline. It reduces mismatch between the task and the output layer, since paraphrase detection is fundamentally binary rather than open-vocabulary generation. It also removes unnecessary computation: the baseline first computes scores for the entire vocabulary, even though only two token ids matter during evaluation.

We further improved the training pipeline by adding dropout, weight decay, gradient clipping, mixed-precision training, and a linear warmup-plus-decay learning-rate schedule. Let T be the total number of training steps and T_w the number of warmup steps. At step t , the learning rate is

$$\eta_t = \begin{cases} \eta \cdot \frac{t+1}{T_w}, & t < T_w, \\ \eta \cdot \max\left(0, 1 - \frac{t-T_w}{T-T_w}\right), & t \geq T_w. \end{cases} \quad (15)$$

We also clip the gradient norm to a maximum of 1.0. These changes are not novel algorithms, but they make fine-tuning substantially more stable and improve downstream accuracy in practice.

Finally, we added explicit label normalization so that labels represented either as $\{0, 1\}$ or as the GPT-2 token ids for no and yes are mapped into a consistent target format before training. This eliminates a brittle implementation detail in the starter pipeline and makes the training loop more robust.

4.4 Sonnet Generation

For sonnet generation, we return GPT-2 to its native autoregressive setting. The model is conditioned on a prefix consisting of the first three lines of a Shakespearean sonnet and is trained to generate the remaining continuation token by token. Unlike the classification tasks, this setting does not require an additional task-specific head. Instead, it directly uses the decoder-only language modeling capability of GPT-2.

Given a prompt prefix p and target continuation tokens $y_1, \dots, y_m$, the training objective is the conditional language modeling loss

$$\mathcal{L}_{\text{sonnet}} = - \sum_{t=1}^m \log p(y_t \mid p, y_{<t}). \quad (16)$$

This is the natural downstream objective for GPT-2, since it matches the model’s original pretraining formulation.

At inference time, generation proceeds autoregressively. Starting from the three-line prompt, the model predicts the next token, appends it to the current context, and repeats until the continuation is complete. This process allows the model to condition each newly generated token on both the initial prompt and all previously generated content.

We evaluate sonnet generation with chrF, a character n -gram F-score commonly used in automatic text generation evaluation. chrF provides a simple quantitative measure of overlap between the generated continuation and the reference continuation. While this metric does not fully capture poetic quality, style, or thematic coherence, it offers a consistent and reproducible way to compare generation performance across runs.

Methodologically, the sonnet task complements the sentiment and paraphrase tasks in two ways. First, it tests whether our GPT-2 implementation functions as a true autoregressive language model rather than only as a feature extractor for classification. Second, it allows us to examine whether improvements in the shared architecture and optimization pipeline transfer not only to discriminative prediction but also to multi-token text generation.

5 Experiments

5.1 Data

We used the datasets provided in the project repository: SST (train 8,544 / dev 1,101 / test 2,210), CFIMDB (train 1,701 / dev 245 / test 488), Quora paraphrase subset (train 141,506 / dev 20,215 / test 40,431), and Shakespeare sonnets (train 143 / test 12 in the handout description). Preprocessing included GPT-2 BPE tokenization; truncation/padding to a fixed max length; label mapping for sentiment and paraphrase; cloze prompt formatting for Quora; and preserving line breaks for sonnet generation. We trained only on the provided train/dev splits and kept the test sets strictly for final evaluation/submission.

5.2 Evaluation method

We evaluated model performance using the provided development splits for each task. For sentiment analysis and paraphrase detection, the model generated predictions on the full dev datasets, which were then formatted according to the project submission requirements. For sonnet generation, the model generated 12 sonnets following the given first 3 lines of the sonnets.

All dev predictions were submitted to the Gradescope autograder to obtain standardized performance score. We used the resulting scores and qualitative inspection of outputs to guide model tuning, including adjustments to hyperparameters, training configurations, and generation settings. The held-out test sets were not used during model development and were reserved strictly for final evaluation and submission.

5.3 Experimental details

5.3.1 Paraphrase Detection

We fine-tuned pretrained GPT-2 for Quora paraphrase detection using the assignment’s cloze-style formulation, where the model predicts the target tokens “yes” or “no” for each sentence pair. Our main model uses the final-token hidden state from GPT-2 with a lightweight linear classification head, and is trained end-to-end with AdamW, mixed-precision GPU training, gradient clipping, and a linear warmup/decay learning-rate schedule. The strongest configuration remained close to the original working setup: GPT-2 base, batch size 8, 6 epochs, learning rate $8e-6$, dropout 0.1, weight decay 0.01, max grad norm 1.0, and 6% warmup. We selected checkpoints by development accuracy.

We also ran a more aggressive optimization variant intended to improve speed, but it produced worse development performance. In that version, we replaced the original yes/no token-level training objective with a direct 2-class objective, introduced a deeper MLP classification head, added label smoothing and differential learning rates, and shortened training to fewer epochs. In hindsight, this experiment changed too many factors simultaneously and moved too far away from the assignment’s original supervision format. The main mistake was not isolating variables: the performance drop could have come from the objective change, the extra regularization, or reduced training time. By

contrast, the improvements that were correct and retained were low-risk engineering optimizations that did not alter the task formulation, such as removing unnecessary full-vocabulary computation during training, using mixed precision, enabling TF32 where supported, and improving dataloader efficiency. Overall, this ablation process showed that preserving the original yes/no token supervision and simple head architecture was more reliable than introducing multiple optimization changes at once.

5.3.2 Sonnet Generation

We fine-tuned the model for the sonnet generation task using a partial fine-tuning strategy rather than full fine-tuning. Specifically, we unfroze only the last K transformer layers while keeping all lower layers frozen. Empirically, K=1 achieved the best balance between adaptation and generalization, whereas unfreezing more layers led to rapid overfitting due to the small size of the sonnet dataset.

To monitor training dynamics, we created an internal train/validation split from the provided training sonnets. A 9:1 split yielded the most stable validation behavior and enabled consistent model selection across experiments. We also implemented early stopping based on average token-level validation loss, but this mechanism did not effectively prevent overfitting. In practice, the loss metric did not sufficiently capture qualitative degradation in generated text. However, it still provided a useful reference signal: we observed a consistent relationship between validation loss values and the autograder performance, allowing us to estimate the approximate loss range corresponding to the best evaluation scores. This heuristic helped guide the selection of optimal epochs and learning rates when testing new strategies.

To better align generation with the structural characteristics of Shakespearean sonnets, we incorporated sonnet-aware decoding constraints. Constraining the approximate word count per line improved rhythmic consistency and readability, while explicitly limiting the total number of generated lines did not lead to performance improvements and was therefore not adopted in the final configuration. We also experimented with restricting generated tokens to English letters and common punctuation symbols, but this constraint reduced expressive flexibility and occasionally disrupted natural token transitions learned during pretraining, resulting in less coherent outputs.

We further improved generation quality by enhancing the decoding strategy. In addition to nucleus sampling (top-p), we incorporated top-k sampling and found that the combination of top-k = 50 and top-p = 0.95 consistently produced the most coherent and stylistically appropriate outputs. To mitigate repetitive phrasing, we applied no_repeat_ngram_size = 3, which effectively reduced local repetition and improved fluency. We also generated num_return_sequences = 4 candidate continuations for each prompt and selected the highest-quality sample based on its average log-probability. Overall, these adjustments reduced repetitive loops and degenerate text, while improving diversity and robustness by avoiding poor local choices that can arise from relying on a single stochastic sample.

Finally, we adopted a learning rate scheduling strategy that used a relatively larger learning rate during early epochs to accelerate adaptation, followed by gradual decay for stabilization. For each architectural or decoding adjustment, we conducted targeted sweeps over learning rate and training epochs, using validation loss trends and qualitative inspection of generated sonnets to identify the best configuration.

5.4 Results

We report the final model performance on each task as evaluated by the Gradescope autograder.

Task	Metric / Setting	Score
Sentiment Analysis	SST Dev Accuracy (Last Linear Layer)	0.467
	SST Dev Accuracy (Full Model)	0.539
	CFIMDB Dev Accuracy (Last Linear Layer)	0.861
	CFIMDB Dev Accuracy (Full Model)	0.980
Paraphrase Detection	Dev Accuracy	0.890
	Test Accuracy	0.861
Sonnet Generation	Dev chrF Score	42.282
	Test chrF Score	41.917

6 Analysis

Our results show that transfer from a pretrained decoder-only language model is effective, but highly sensitive to task formulation and fine-tuning strategy. Across all experiments, the strongest performance came from methods that stayed close to GPT-2’s pretrained objective while adding only lightweight task-specific adaptation.

For sentiment analysis, the contrast between SST and CFIMDB is especially informative. On CFIMDB, full-model fine-tuning improved dev accuracy steadily from 0.951 at epoch 0 to a peak of 0.980 at epoch 5, showing that end-to-end adaptation works well for binary review classification with longer, richer inputs. On SST, full-model fine-tuning improved dev accuracy from 0.341 at epoch 0 to 0.511 at epoch 4, then declined slightly even as training accuracy kept increasing. This pattern suggests overfitting and indicates that the smaller, fine-grained five-class setting is harder to optimize with full decoder-only adaptation. In this case, freezing most of the pretrained backbone appears to provide useful regularization.

For paraphrase detection, our main finding is that simple, task-aligned adaptation was more reliable than aggressive modification. The strongest model kept the original yes/no prompt formulation and used a lightweight head with stable optimization. Its dev accuracy improved consistently across training, from 0.8559 at epoch 0 to 0.8898 at epoch 5, with dev F1 rising from 0.8494 to 0.8830. This smooth trajectory suggests that preserving the original cloze-style structure mattered more than adding architectural complexity. A more aggressive variant with a deeper head, altered supervision, and extra regularization performed worse, reinforcing the importance of changing one factor at a time.

For sonnet generation, we observed that full-model fine-tuning led to rapid overfitting due to the small size of the poetry corpus. Before freezing any layers, the best performance achieved was a score of 39.777. Partial fine-tuning provided a more favorable balance between adaptation and generalization: unfreezing the last two transformer layers improved the score to 41.953, while updating only the last layer further increased performance to 42.282 (best epoch 9), indicating that minimal adaptation at the top of the network was sufficient for effective stylistic transfer. After introducing an adaptive learning rate schedule, the best-performing checkpoint shifted to a later stage (epoch 25), suggesting that slower late-stage optimization helped stabilize training in this low-resource generative setting. Decoding strategy also played a critical role in output quality. Introducing a soft constraint on approximate word count per line improved rhythmic consistency and increased the best score from 41.839 to 41.953. More generally, the strongest sampling configurations involved relatively high values of top-p, top-k, and temperature, reflecting the inherently creative and diverse nature of poetic generation. These findings indicate that for structured creative text generation, lightweight parameter adaptation combined with high-diversity decoding strategies can substantially improve fluency and stylistic expressiveness.

Overall, our analysis shows that implementation fidelity and task alignment were the main drivers of performance. Better results did not come from making the model more complex, but from choosing adaptation strategies that respected the strengths and limitations of a pretrained autoregressive Transformer.

7 Conclusion

In this project, we implemented the core GPT-2 architecture and AdamW optimizer, loaded pretrained weights, and adapted the same decoder-only backbone to sentiment classification, paraphrase detection, and sonnet generation. Our results show that a single pretrained language model can support both discriminative and generative NLP tasks when the downstream formulation is well aligned with the model’s training objective.

Our main findings are threefold. First, transfer performance depends strongly on the task: full-model fine-tuning helped on CFIMDB and on SST. Second, for paraphrase detection, a simple and stable prompt-based setup outperformed more aggressive architectural changes, leading to a final test accuracy of 0.861. Third, for sonnet generation, partial fine-tuning and lightweight generation control improved output quality and produced a test chrF score of 41.917, slightly above the baseline.

The main limitation of our work is that some tasks, especially sonnet generation, are constrained by limited data and are therefore sensitive to overfitting. In addition, our qualitative analysis for generation is necessarily limited by automatic evaluation, which does not fully capture style or literary quality. Future work could explore stronger decoding strategies, more systematic ablations, and richer qualitative evaluation of generated outputs. Overall, this project gave us a concrete understanding of how pretrained autoregressive Transformers transfer across tasks, and how much careful implementation and tuning matter in practice.

Team contributions

- Yuchan Guo: Implemented the Adam optimizer and performed fine-tuning experiments for the paraphrase detection task.
- Yushi Feng: Implemented the GPT-2 model and conducted fine-tuning experiments for sonnet generation.

All remaining components of the project not explicitly listed above were shared and completed equally between both team members.

References

- ¹Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., Polosukhin, I. (2017, June 12). Attention is all you need. arXiv.org. <https://arxiv.org/abs/1706.03762>
- ²Radford, A., Sutskever, I., Amodei, D., Luan, D., Child, R., Wu, J. (n.d.). Language models are unsupervised multitask learners. https://cdn.openai.com/better-language-models/language_models_are_unsupervised_multitask_learners.pdf
- ³Wang, A., Singh, A., Michael, J., Hill, F., Levy, O., Bowman, S. (n.d.). Glue: A multi-task benchmark and analysis platform for natural language understanding. ACL Anthology. <https://aclanthology.org/W18-5446/>
- ⁴Devlin, J., Chang, M.-W., Lee, K., Toutanova, K. (n.d.). Bert: Pre-training of deep bidirectional Transformers for language understanding. ACL Anthology. <https://aclanthology.org/N19-1423/>
- ⁵Schick, T., Schütze, H. (2021). Exploiting cloze-questions for few-shot text classification and Natural Language Inference. Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: Main Volume, 255–269. <https://doi.org/10.18653/v1/2021.eacl-main.20>
- ⁶Kingma, D. P., Ba, J. (2014, December 22). Adam: A method for stochastic optimization. arXiv.org. <https://arxiv.org/abs/1412.6980>
- ⁷Lau, J. H., Cohn, T., Baldwin, T., Brooke, J., Hammond, A. (2018). Deep-Speare: A joint neural model of poetic language, meter and rhyme. Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), 1948–1958. <https://doi.org/10.18653/v1/p18-1181>
- ⁸Radford, A., Sutskever, I., Salimans, T., Narasimhan, K. (n.d.). Improving language understanding by generative pre-training Alec Radford openai. https://cdn.openai.com/research-covers/language-unsupervised/language_understanding_paper.pdf