

Building GPT-2 for Paraphrase Detection and Sonnet Generation

CS224N Default Final Project Report (Winter 2026)

Yifan Guo
Stanford University
yifanguo@stanford.edu

1 Key Information

- Team member names and emails: Yifan Guo (yifanguo@stanford.edu)
- Project type: Default final project
- Sharing project with another class: No
- Mentor: Ahmed Ahmed
- External collaborators: None

Abstract

This report describes my implementation and evaluation of the CS224N default final project based on GPT-2 Radford et al. [2019], dfp. I completed the missing GPT-2 components in the starter code, validated the implementation with the provided optimizer and sanity tests, and evaluated the resulting system on sentiment classification, paraphrase detection, and sonnet generation. A remote Tesla T4 run reproduced the sentiment baselines above the handout references, and the strongest completed paraphrase configuration reached 0.888 Quora dev accuracy with tuned thresholding. The final selected sonnet configuration reaches 41.67 chrF on dev and 41.683 chrF on the test leaderboard. The final test leaderboard results are 0.879 paraphrase accuracy and 41.683 sonnet chrF. These paraphrase results indicate that a carefully tuned plain GPT-2 pipeline with a learned binary head can already perform strongly without requiring a larger backbone.

2 Introduction

The CS224N default final project asks students to build GPT-2 from partially completed code and then apply the resulting model to three downstream tasks: sentiment classification, sentence-pair paraphrase detection, and sonnet generation dfp. The project is a useful testbed because it combines two different forms of language-model transfer. Sentiment classification and paraphrase detection evaluate whether the model can produce task-relevant representations from pretrained contextual embeddings, while sonnet generation evaluates whether the same decoder can support coherent multi-token generation under a structured literary form.

These tasks are challenging for a small GPT-2 implementation for different reasons. Sentiment classification is the most straightforward because the label space is fixed and the supervision signal is direct. Paraphrase detection is harder because the model must encode sentence-pair meaning into a representation that supports a clean binary classification decision. Sonnet generation is harder still because the training set is tiny, long-form decoding compounds errors, and quality is judged by an n-gram overlap metric that only partially reflects fluency and poetic quality.

My open-ended effort on the paraphrase side ultimately centered on the extensions explicitly suggested in Section 7 of the handout: broader hyperparameter optimization, stronger regularization,

and checkpoint ensembling dfp. In practice, I fully implemented the default GPT-2 stack, ran paraphrase tuning experiments with dev-only model selection, and analyzed which handout-supported improvements were practical under limited compute. The main contribution of this report is therefore a complete implementation of the default project, a careful evaluation on the required tasks, and an analysis of which paraphrase improvements were most effective within the available budget.

3 Related Work

GPT-style language models showed that large decoder-only transformers can acquire broad linguistic knowledge through autoregressive next-token prediction and then transfer to downstream tasks via fine-tuning or prompting Radford et al. [2019]. The CS224N default project uses this paradigm directly: sentiment is treated as classification from the final hidden state, paraphrase detection is treated as sentence-pair classification using a serialized pair input, and sonnet generation uses standard left-to-right language modeling dfp.

For generation, nucleus sampling is a standard decoding strategy for reducing the degeneration often seen in pure greedy or beam-style decoding, especially for open-ended text generation Holtzman et al. [2020]. The provided sonnet code already exposes temperature and top- p as controllable decoding parameters, making it natural to study decoding trade-offs even without changing the underlying model architecture.

For paraphrase detection specifically, the handout recommends several natural improvement directions beyond the plain baseline: larger models, different optimizers and regularization, hyperparameter optimization, ensembling, and PEFT-style methods dfp. The work reported here focuses on the subset I implemented and tested directly in this repository: hyperparameter sweeps, dropout and weight-decay tuning, checkpoint ensembling, and an exploratory GPT-2-medium + LoRA path. The selected final result still comes from the stronger plain GPT-2 configuration, while the LoRA experiments are kept as a larger-model ablation that did not outperform the selected GPT-2 pipeline under the available T4 budget Hu et al. [2022].

4 Approach

Default GPT-2 baseline. The core model follows the provided GPT-2 setup from the official handout dfp. I implemented masked self-attention, GPT-2 transformer blocks, tied token embeddings, the AdamW optimizer, sentiment classification, paraphrase prediction, and autoregressive sonnet generation inside the starter repository. The benchmark throughout the report is this plain default GPT-2 setup before the open-ended paraphrase improvements described below. The starter scaffolding comes from the official CS224N release, while the missing GPT-2 components and task-specific additions described below were implemented in this project unless otherwise cited.

At the transformer level, each layer applies masked self-attention followed by an MLP block with residual connections and layer normalization. For a hidden-state matrix X , the attention computation is

$$Q = XW_Q, \quad K = XW_K, \quad V = XW_V, \\ \text{Attn}(X) = \text{softmax}\left(\frac{QK^\top}{\sqrt{d_k}} + M\right)V,$$

where M is the causal mask that prevents each position from attending to future tokens. Implementing this correctly was necessary for both the classification tasks and sonnet generation, since any masking bug would leak future information during training.

For sentiment classification, given tokens $x_{1:T}$, GPT-2 produces hidden states $h_{1:T} = f_\theta(x_{1:T})$. The classifier uses the last-token representation h_T and predicts

$$\hat{y} = W_s h_T + b_s, \quad \mathcal{L}_{\text{sent}} = - \sum_{c=1}^C y_c \log(\text{softmax}(\hat{y})_c).$$

For paraphrase detection, the input for example i is serialized as

Is “ s_1 ” a paraphrase of “ s_2 ”? Answer “yes” or “no”:

The prompt is only the textual packaging of the sentence pair. The decision itself is made by a learned binary head on the final GPT-2 representation:

$$z_i = W_p h_T + b_p = [z_{i,\text{no}}, z_{i,\text{yes}}].$$

The paraphrase probability is

$$p_i(\text{yes}) = \frac{\exp(z_{i,\text{yes}})}{\exp(z_{i,\text{yes}}) + \exp(z_{i,\text{no}})}, \quad \hat{y}_i = \mathbf{1}[p_i(\text{yes}) \geq \tau].$$

For training, labels are binary ($y_i \in \{0, 1\}$) and optimized with cross-entropy on these two logits. The constants 8505 (*yes*) and 3919 (*no*) are only used when serializing CSV predictions for the autograder format dfp; they are not trainable parameters. The selected final paraphrase pipeline in this report is therefore: plain GPT-2 backbone, serialized sentence-pair input, a learned binary classification head on the last hidden state, AdamW optimization, dropout and weight-decay tuning, and dev-only threshold selection.

Paraphrase improvements from Section 7. The open-ended paraphrase work focuses on three improvements suggested directly by the handout dfp. First, I run broader hyperparameter optimization over learning rate, sequence length, train coverage, and batch size. Second, I tune regularization through AdamW weight decay and transformer dropout. The optimizer remains AdamW, but the decoupled weight-decay term becomes an explicit tuning knob:

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

Third, I ensemble the strongest independently trained checkpoints by averaging their predicted paraphrase probabilities:

$$\bar{p}_i(\text{yes}) = \frac{1}{M} \sum_{m=1}^M p_i^{(m)}(\text{yes}), \quad \hat{y}_i = \mathbf{1}[\bar{p}_i(\text{yes}) \geq \tau].$$

All paraphrase model selection is done strictly on the train/dev splits. Test predictions are exported only after selecting the final checkpoint or ensemble on dev.

To probe the larger-model direction from Section 7, I also implemented an exploratory GPT-2-medium + LoRA setup. If a pretrained linear map is W , LoRA replaces it at training time with

$$W' = W + \Delta W, \quad \Delta W = BA, \quad \text{rank}(\Delta W) = r,$$

where $A \in \mathbb{R}^{r \times d_{\text{in}}}$ and $B \in \mathbb{R}^{d_{\text{out}} \times r}$ are the trainable low-rank factors Hu et al. [2022]. In my implementation, the base GPT-2-medium weights are frozen and only the LoRA factors plus the binary paraphrase head are optimized. These runs were useful for testing whether a larger backbone could fit comfortably on a T4, but the strongest completed dev result still came from the plain GPT-2 system.

For sonnet generation, GPT-2 is fine-tuned as an autoregressive language model over full sonnets. At decoding step t ,

$$p(w_t \mid w_{<t}) = \text{softmax}\left(\frac{z_t}{\tau}\right),$$

and the implementation supports nucleus sampling from the minimal token set V_p whose cumulative mass reaches top- p Holtzman et al. [2020].

Sonnet generation approach. The sonnet task uses the same GPT-2 backbone in a pure generation setting rather than a classification setting. During fine-tuning, each Shakespeare sonnet is treated as one autoregressive training sequence with its original line-break structure preserved, so the model learns both local token transitions and the broader stylistic rhythm of the corpus. At evaluation time, generation is conditioned on the first three lines of a held-out sonnet prompt, and the model decodes the remaining continuation autoregressively until the full poem is completed. The main sonnet-side tuning decisions are therefore not architectural changes, but checkpoint selection and decoding control: I compare temperature and top- p settings on the held-out prompts, then keep the configuration that gives the best chrF while still producing qualitatively coherent continuations.

Optimizer and implementation details. The starter project also requires an AdamW implementation. For parameters w_t , gradient g_t , first-moment state m_t , and second-moment state v_t , the update is

$$\begin{aligned} m_t &= \beta_1 m_{t-1} + (1 - \beta_1) g_t, & v_t &= \beta_2 v_{t-1} + (1 - \beta_2) g_t^2, \\ \hat{m}_t &= \frac{m_t}{1 - \beta_1^t}, & \hat{v}_t &= \frac{v_t}{1 - \beta_2^t}, \\ w_t &= w_{t-1} - \eta \left(\frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon} + \lambda w_{t-1} \right). \end{aligned}$$

Passing the provided optimizer test and sanity check was important for establishing that later downstream results reflect task difficulty rather than a broken GPT-2 implementation.

5 Experiments

Data. I use the datasets bundled with the default project dfp: SST (train/dev/test: 8544/1101/2210), CFIMDB (1701/245/488), Quora paraphrase (141506/40429/40431 in the provided repository files), and Shakespeare sonnets (143 training sonnets plus 12 held-out prompts for development-style generation).

Evaluation method. The handout uses accuracy for SST, CFIMDB, and Quora paraphrase, and chrF for sonnet generation Popović [2015], dfp. I follow the same metrics. For paraphrase detection, the decision threshold τ is tuned on dev only, and the final report separately records the required paraphrase and sonnet leaderboard test scores.

Experimental details. The implementation was validated locally with `optimizer_test.py` and `sanity_check.py`, both of which pass in the current repository. The sentiment numbers reported here come from a complete GPU base run on a remote Tesla T4 instance, using dataset-specific batch sizes and separate hyperparameters for last-linear-layer versus full-model fine-tuning. The selected paraphrase development number comes from a longer GPT-2 paraphrase run on the same T4 setup with a learned binary head, 3 epochs, $lr = 5 \times 10^{-5}$, weight decay 10^{-2} , hidden/attention dropout 0.05, max length 128, and tuned decision threshold. The sonnet development number comes from the final selected generation configuration evaluated on the held-out prompts, where each prompt provides the first three lines and the model generates the continuation under temperature/top- p decoding.

I also wrote helper scripts for launching experiments, collecting metrics, and packaging prediction files in the required format. These scripts reduced bookkeeping errors and made it easier to compare model variants consistently. The code submission packaged for Gradescope excludes data and checkpoints, in line with the official report instructions.

The most important practical constraint on these experiments was compute and wall-clock budget. Early debugging and smoke testing were done locally without a CUDA GPU, then the final sentiment and paraphrase reproduction runs were moved to a remote Google Cloud T4 instance once the pipeline was stable. Limited GPU budget still constrained the breadth of the paraphrase hyperparameter search.

Stage	Key hyperparameters	Selected values
Sentiment (last-linear)	epochs, lr, batch size (SST/CFIMDB)	10, 1×10^{-3} , 64 / 8
Sentiment (full-model)	epochs, lr, batch size (SST/CFIMDB)	10, 1×10^{-5} , 16 / 8
Paraphrase (best completed run)	epochs, lr, batch size, max length, threshold τ	3, 5×10^{-5} , 64, 128, 0.653
Sonnet (final selected)	epochs, lr, batch size, decoding T , top- p	5, 1×10^{-5} , 8, 1.2, 0.9

Table 1: Main hyperparameters used for the reported runs.

6 Results

Table 2 summarizes the strongest verified development results in the repository. The sentiment values are full GPU reproductions from the base pipeline, the paraphrase value is from a completed longer GPU run, and the sonnet value is from the final selected generation configuration.

Task / Metric	Reference	Local Result	Notes
SST dev acc (last-linear-layer)	0.462	0.470	T4 GPU base run
SST dev acc (full-model)	0.513	0.517	T4 GPU base run
CFIMDB dev acc (last-linear-layer)	0.861	0.865	T4 GPU base run
CFIMDB dev acc (full-model)	0.976	0.988	T4 GPU base run
Quora dev acc	N/A	0.888	GPT-2 T4 run + tuned threshold
Sonnet dev chrF	N/A	41.67	final selected generation setup

Table 2: Verified results. Reference values are the handout baselines for the sentiment tasks dfp.

These results support three conclusions. First, the core GPT-2 implementation is functionally correct enough to reproduce all four published sentiment references, which is stronger evidence than the earlier CPU smoke tests. Second, the sentiment and paraphrase pipelines needed practical adjustments beyond the starter defaults, especially dataset-specific batch sizes, stable GPU execution, and paraphrase threshold tuning. Third, sonnet quality is competitive under a carefully selected generation setup, with chrF above the target range.

The sentiment numbers are best interpreted as the current base-model benchmark in this repository. They show that both last-layer and full-model fine-tuning modes execute end-to-end, save all eight required CSV files, and meet or slightly exceed the handout references on SST and CFIMDB. The longer paraphrase GPT-2 run lifts Quora dev performance to 0.888 with a tuned threshold $\tau \approx 0.653$, and the corresponding final test leaderboard score reaches 0.879. These results suggest that the selected plain GPT-2 checkpoint is already strong after focused hyperparameter search and longer GPU training. Exploratory GPT-2-medium + LoRA runs were technically successful on the T4 but did not surpass the selected plain GPT-2 checkpoint, so they remain an ablation rather than the final chosen system. On sonnet generation, the selected setup reaches 41.67 chrF on dev and 41.683 chrF on test.

Table 3 lists the final test leaderboard scores used for grading, as requested in the course handout dfp.

Required Final Score	Value
Quora paraphrase test leaderboard accuracy	0.879
Sonnet generation test leaderboard chrF	41.683

Table 3: Final test leaderboard scores reported for the default-project grading requirement.

7 Analysis

The paraphrase system still shows a recognizable failure pattern: it can confuse surface similarity with semantic equivalence. A representative false positive is a pair of long questions about bars affiliated with sports teams, where the sentence structure is almost identical but the city is changed from San Francisco to Atlanta. Another false positive swaps the target programming language from Haskell to Python while preserving nearly the entire question template. These examples show that lexical overlap remains a strong confounder whenever a small entity substitution changes semantic intent.

False negatives have the opposite structure. When two questions are true paraphrases but differ substantially in wording, the model becomes too conservative. Examples include questions about jumping through a hole in the Earth and about explaining how creationists reconcile a 6,000-year-old universe with ancient starlight. In both cases, the semantics are close, but wording divergence still suppresses the predicted probability.

The sonnet system has a different failure mode. Outputs tend to preserve line length, diction, and occasional rhyme, which helps chrF, but novelty can still be limited. Some continuations follow the prompt style strongly yet remain conservative in semantic progression. This indicates that checkpoint selection and decoding exploration remain important even when metric performance is already competitive.

Error Type	Example A	Example B
False positive	sports bars in San Francisco	sports bars in Atlanta
False positive	Haskell interview question	Python interview question
False negative	jump through a hole in Earth	drill through Earth and jump in
False negative	creationist universe question	starlight older than 6,000 years question

Table 4: Representative Quora dev errors. Small entity substitutions create false positives, while larger paraphrastic rewrites create false negatives.

Table 5 shows a compact example. The first three prompt lines come from a held-out sonnet, and the continuation remains stylistically coherent in Shakespearean diction with strong form consistency.

Held-out prompt + continuation excerpt
<i>Prompt:</i> “Thine eyes I love, and they, as pitying me / Knowing thy heart torment me with disdain / Have put on black, and loving mourners be.”
<i>Continuation excerpt:</i> “With lines and wrinkles; when his youthful morn / Hath traveled on to age’s steepy night / And all those beauties whereof now he’s king ...”

Table 5: A representative dev-set sonnet output from the selected final configuration.

From an engineering perspective, one useful outcome of this project is the separation between *model quality* and *submission robustness*. Helper scripts guarantee valid files for the dev and test leaderboards, while the main GPT-2 codebase remains the correct place to pursue modeling improvements. That separation reduced deadline risk and made later GPU iterations easier to manage.

8 Limitations and Remaining Work

The main limitation of the current report is compute budget. The paraphrase improvements that seem most promising still require more GPU time than I had available for a broader search. The handout-aligned extensions I implemented and tested are hyperparameter sweeps, regularization tuning, ensembling, and an exploratory GPT-2-medium + LoRA path, but the search remains narrower than ideal. In particular, the available GPU budget limited broader multi-seed sweeps, longer schedules, and more extensive larger-model trials.

Despite those limitations, the repository is in a materially better state than it was at the proposal stage. The core implementation is complete, the downstream scripts run, prediction files are generated in the expected format, and the report sources compile. In practical terms, the project now has a stable foundation for stronger GPU-backed revision in the future.

9 Conclusion

This project completed the core GPT-2 implementation for the CS224N default final project and evaluated it on sentiment classification, paraphrase detection, and sonnet generation. The strongest verified results include full GPU sentiment reproduction above the handout references, 0.888 paraphrase dev accuracy with 0.879 test accuracy, and 41.67/41.683 sonnet dev/test chrF. The main lesson is that careful implementation and focused GPU experimentation can deliver a strong final system, and that the selected plain GPT-2 paraphrase pipeline already performs well without requiring a larger backbone.

The next steps are clear. First, continue paraphrase sweeps that stay strictly on the train/dev splits during model selection. Second, push the handout-supported paraphrase extensions further with longer schedules, broader regularization search, larger-model trials if memory allows, and checkpoint ensembling across multiple seeds. Third, continue sonnet checkpoint and decoding selection to test whether additional improvements are possible beyond the current 41.683 chrF test score.

References

- 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.
- Cs224n default final project: Build gpt-2 (winter 2026 handout). https://web.stanford.edu/class/cs224n/project/DFP_Instructions.pdf. Accessed 2026-02-25.
- Ari Holtzman, Jan Buys, Li Du, Maxwell Forbes, and Yejin Choi. The curious case of neural text degeneration. In *International Conference on Learning Representations*, 2020.
- 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. In *International Conference on Learning Representations*, 2022.
- 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, 2015.