

CS 224N Default Project

Stanford CS224N {Default} Project

Yaojing Huang He

Department of Civil and Environmental Engineering
Stanford University
hhyj4495@stanford.edu

Kayla Li

Department of Management Science & Engineering
Stanford University
peizheli@stanford.edu

Abstract

In this project, we implement a GPT-2 style decoder-only Transformer. The model is first validated on SST and CFIMDB benchmarks, after which we apply it to two downstream tasks: paraphrase detection and Shakespeare-style sonnet generation. For paraphrase detection, we incorporate LoRA with SMART regularization, while for sonnet generation we apply Direct Preference Optimization (DPO). Our models outperform the staff baselines, achieving 0.883 test accuracy on paraphrase detection and 41.864 chrF on sonnet generation.

1 Key Information to include

- Mentor: Nevin George
- External Collaborators (if you have any): No
- Sharing project: No
- Late Days (if you are using any):

Team member	Late days remaining
<Kayla Li>	<6>
<Yaojing Huang He>	<6>
Total pooled	<12>
Team size	<2>
Late days applied for the report	<0>

2 Introduction

Decoder-only Transformers trained with next-token prediction can transfer to many tasks through fine-tuning. GPT-2 is a classic example of this approach [1]. The CS224N default project makes this concrete by asking us to implement key GPT-2 components and then fine-tune the model for multiple downstream tasks.

In this project, we first validate our implementation of GPT-2 by evaluating it on several classification benchmarks, including *SST last-linear-layer*, *SST full-model*, *CFIMDB last-linear-layer*, and *CFIMDB full-model*. These tasks serve as sanity checks to verify that our model architecture, training pipeline, and optimization procedures behave as expected.

After establishing a working GPT-2 implementation, we apply the model to two downstream tasks: paraphrase detection and sonnet generation. The goal of the paraphrase task is to determine whether

two input sentences express the same meaning, while the sonnet task aims to generate coherent Shakespeare-style sonnet continuations.

Beyond running the baseline models, we further explore methods to improve performance under practical constraints. For the sonnet generation task, we found this evaluation is highly sensitive to decoding behavior, meaning that training and decoding must be considered jointly. Therefore, we apply Direct Preference Optimization (DPO) to encourage the model to assign higher probability to preferred continuations and avoid degenerate or low-quality generations without relying on RLHF. For the paraphrase detection task, we adopt parameter-efficient fine-tuning with LoRA and incorporate SMART regularization to reduce overfitting and improve training stability.

3 Related Work

Our implementation follows the Transformer architecture [2] and GPT-2 [1]. We optimize with AdamW, which combines Adam [3] and decoupled weight decay [4].

For the paraphrase detection task, LoRA is a widely used parameter-efficient method that trains low-rank adapter matrices instead of updating all model weights [5]. SMART regularization improves robustness by penalizing prediction changes under worst-case embedding perturbations [6].

For the sonnet generation task, a common approach to align a language model with human preferences is RLHF. RLHF typically trains a reward model from pairwise comparisons and then optimizes the policy with reinforcement learning (often PPO) under a KL constraint to a reference model. This pipeline is effective but can be unstable and expensive to tune.

Direct Preference Optimization (DPO) simplifies this setup by optimizing a supervised objective on preference pairs while still using a fixed reference model [7]. DPO can be derived as an equivalent form of KL-regularized RLHF under standard assumptions. In practice, it removes the explicit reward-model training stage and avoids running PPO, while keeping the key signal: the model should assign higher probability to preferred outputs than to rejected outputs.

4 Approach

We follow the default-project pipeline: implement GPT-2 small (decoder-only Transformer) and AdamW from the scaffold, then fine-tune on SST/CFIMDB (classification), Quora (cloze-style “yes/no” paraphrase), and Shakespeare sonnets (next-token prediction + conditional generation).

Baseline. Our baseline is the official CS224N starter code (`cs224n_gpt`) and its default training/evaluation recipes; we compare against staff-reported baselines for sentiment and leaderboard metrics. The underlying model is standard GPT-2 [1] with masked multi-head self-attention [2] and AdamW [3, 4].

What we implemented for the Baseline We completed: (i) causal self-attention (`modules/attention.py`), (ii) GPT-2 block forward/residual logic (`modules/gpt2_layer.py`), (iii) embeddings + weight tying (`models/gpt2.py`), (iv) sentiment classifier head (`classifier.py`), (v) per-token logits for sonnet LM training (`sonnet_generation.py:SonnetGPT.forward`), (vi) and forward module for paraphrase detection (`ParaphraseGPT.forward`).

Sonnet generation with DPO For the sonnet generation task, we first established a baseline using **GPT-2 Small**. To improve sonnet generation, we introduced a simple decoding refinement by adding top- k filtering before nucleus sampling (top- p) during generation; we will quantify its effect via controlled decoding sweeps and add training stabilization (gradient clipping, early stopping) in follow-up experiments.

We treat sonnet continuation as a preference optimization problem. Each training example is a triple (x, y_w, y_l) , where x is the prompt (the first three lines), y_w is a preferred continuation, and y_l is a dispreferred continuation. We optimize a policy model π_θ relative to a fixed reference model π_{ref} using DPO [7]:

$$\mathcal{L}_{\text{DPO}} = -\mathbb{E}_{(x, y_w, y_l)} \left[\log \sigma(\beta(\Delta_\pi - \Delta_{\text{ref}})) \right],$$

$$\Delta_{\pi} = \log \pi_{\theta}(y_w|x) - \log \pi_{\theta}(y_l|x),$$

$$\Delta_{\text{ref}} = \log \pi_{\text{ref}}(y_w|x) - \log \pi_{\text{ref}}(y_l|x).$$

Here $\sigma(\cdot)$ is the sigmoid and β controls preference strength. Intuitively, DPO increases the model’s log-probability margin for the preferred continuation compared to the rejected one, while accounting for the reference model’s margin.

We implement π_{ref} by copying the fine-tuned GPT-2 model at initialization and freezing it (`ref_gpt = deepcopy(gpt); all requires_grad=False`). We compute sequence log-probabilities with `logprob_continuation()`, which (i) tokenizes the concatenated text $x+y$, (ii) runs the model to obtain logits, (iii) applies `log_softmax`, and (iv) sums token log-probabilities only over the continuation region (computed from the prompt length). We then compute Δ_{π} and Δ_{ref} and apply the DPO loss as `-log(sigmoid(beta*(delta_pi - delta_ref)))`.

Pure preference optimization can drift away from the language modeling objective. To stabilize training on the small sonnet dataset, we mix DPO with a continuation-only cross-entropy loss on the preferred continuation:

$$\mathcal{L} = \lambda_{\text{CE}} \mathcal{L}_{\text{CE}} + (1 - \lambda_{\text{CE}}) \mathcal{L}_{\text{DPO}}.$$

In code this corresponds to `loss = lambda_ce * loss_ce + (1-lambda_ce) * loss_dpo`.

We also performed systematic hyperparameter tuning over learning rate, number of epochs, and decoding parameters, which turned out to be important for achieving strong chrF scores on this small dataset.

Paraphrase Detection For the paraphrase detection task, we first established a baseline using **GPT-2 Small** trained with the standard cross-entropy loss. Specifically, the input sequence is passed through the previously implemented `GPT2Model` within `ParaphraseGPT`. We then use the hidden state of the final token to predict the output token (“yes” or “no”). The hidden representation is fed into the LM head to produce logits over the vocabulary.

On top of this baseline, we explored parameter-efficient fine-tuning using **LoRA**. In our implementation, all original GPT-2 parameters were frozen, including token and position embeddings, all transformer block weights (query, key, value projections, attention/output projections, MLP layers, and layer norms), as well as the final layer norm. During training, only the LoRA adapter parameters attached to the attention *query* and *value* projections in each transformer layer were updated, namely the low-rank matrices `lora_A` and `lora_B`.

In addition, we experimented with **SMART regularization** on the **full GPT-2 Small model**. SMART introduces an adversarial smoothness term into the training objective to improve model robustness. Specifically, we implemented a function `compute_smart_loss(...)` that adds small adversarial perturbations to the input embeddings and computes the KL divergence between the clean logits and the perturbed logits. To support this procedure, we also implemented `forward_with_embeddings(...)` so that the model can run the forward pass directly from embeddings rather than token IDs. The final training objective becomes

$$\text{Loss} = \text{task_loss} + \lambda_{\text{smart}} \cdot \text{smart_loss}$$

where `task_loss` is the standard cross-entropy loss and `smart_loss` encourages the model to produce stable predictions under small embedding perturbations, improving local smoothness and robustness.

We apply SMART in the **full-model setting rather than the LoRA setting** because SMART perturbs the embeddings and propagates gradients through the entire network. Restricting updates only to the small LoRA adapters could limit the effectiveness of the regularization.

Code provenance. We did not write the dataset loaders, most training loops, or leaderboard evaluation; these come from the course starter repository. We use HuggingFace’s GPT-2 tokenizer for preprocessing.

5 Experiments

5.1 Data

We use only the provided CS224N splits: SST (5-class), CFIMDB (binary), Quora paraphrase pairs (binary), and Shakespeare sonnets (143 train; held-out prompts for dev/test).

5.2 Evaluation method

Sentiment is evaluated with dev accuracy (macro-F1 also computed but not our main target). Sonnets are evaluated with Dev/Test chrF [8]. Paraphrase detection is evaluated with Dev/Test accuracy. Comparisons across different training strategies are primarily based on development accuracy.

5.3 Experimental details

5.3.1 Baseline Model

- **Model.** We use GPT-2 as the base architecture. Our baseline model is **GPT-2 Small** (12 layers, $d = 768$, 12 heads) with weight tying for vocabulary logits. For later improvement experiments, we also evaluate larger variants, including **GPT-2 Medium** and **GPT-2 Large**, while keeping the same architecture and training pipeline. Using larger backbone models allows us to test whether performance gains come from increased model capacity or from changes in training strategy.
- **Optimizer.** AdamW (bias correction and decoupled weight decay).
- **Learning rate.** Sentiment (full-model): 10^{-5} to 3×10^{-5} ; last-linear-layer only: 10^{-3} ; Sonnets: 10^{-5} ; Paraphrase: 10^{-5} .
- **Batch size.** SST: 8 (full-model on MPS), up to 64 (last-linear-layer only); CFIMDB: 1–8 (OOM at 8 on MPS in some runs); Sonnets: 2–8; Paraphrase: 32.
- **Epochs.** Sentiment: 1–10; Sonnets: 10–20; Paraphrase: 6–12.
- **Maximum sequence length.** We used the default tokenizer truncation. To reduce the $O(T^2)$ attention and memory cost, we plan to cap CFIMDB sequences (e.g., at 128 or 256 tokens).
- **Hardware and training time.** The following experiments were conducted on an Apple Silicon laptop (MPS): SST (full-model) required ~ 7 – 8 min per epoch ($+ \sim 1$ min for dev evaluation); CFIMDB was slightly slower; Sonnets experiments were conducted on Modal with H100 GPU, requiring ~ 7 – 10 min for 15 epoches. Paraphrase experiments were conducted on an NVIDIA RTX 4080 GPU (16GB VRAM), requiring ~ 15 min per epoch.

5.3.2 Sonnet generation improvement

We performed additional hyperparameter tuning for sonnet generation, focusing on the number of epochs, learning rate, and decoding parameters (temperature and top- p). Because the sonnet dataset is small, performance was highly sensitive to overfitting and decoding randomness. We therefore ran a series of controlled experiments while keeping the model architecture fixed (GPT-2 medium) and only changing one or two hyperparameters at a time.

Experiment	Model	Epochs	LR	Temperature	Top- p	Dev chrF
Exp1	GPT-2 Small	20	1×10^{-5}	1.2	0.95	41.114
Exp2	GPT-2 Medium	10	1×10^{-5}	1.0	0.95	40.857
Exp3	GPT-2 Medium	10	1×10^{-5}	1.5	0.88	41.863
Exp4	GPT-2 Medium	15	1×10^{-5}	1.5	0.88	42.342
Exp5	GPT-2 Medium	15	5×10^{-6}	1.5	0.88	37.680

Table 1: Hyperparameter tuning for sonnet generation. We varied learning rate, decoding temperature, nucleus sampling parameter p , and number of training epochs. The best development chrF was obtained with 15 epochs, default learning rate, temperature 1.5, and top- $p = 0.88$.

Table 1 shows the main hyperparameter configurations we tested for sonnet generation. We found that decoding hyperparameters had a large effect on chrF, often larger than additional training epochs. In

particular, increasing temperature from 1.0 to 1.5 improved diversity and reduced repetitive outputs, while lowering top- p from 0.95 to 0.88 produced more stable line structure. Training for 15 epochs gave the best dev performance, while longer training did not consistently improve results.

We observed that training for too many epochs degraded performance, likely due to overfitting on the small dataset. The best results were obtained at 15 epochs. We also found that slightly higher temperature combined with a smaller top- p value improved fluency while keeping the output close to the reference continuation, which increased chrF.

Constructing preference pairs for DPO (sonnets) The sonnet dataset provides only positive examples (canonical Shakespeare sonnets). We therefore construct dispreferred continuations automatically. We split each training sonnet into a prompt x (first three lines) and a preferred continuation y_w (remaining lines). We generate a dispreferred continuation y_l by corrupting y_w with simple structural perturbations. Concretely, we randomly apply one of: swapping two lines, dropping one line, shuffling a short contiguous block of lines, or shuffling all lines. This produces negatives that are clearly worse in structure while remaining close in surface form. This procedure is implemented in `SonnetsPairDataset` via `_split_prompt_cont()` and `_corrupt()`.

We train with DPO temperature $\beta = \{0.1\}$ and mix coefficient $\lambda_{CE} = \{0.3\}$. We keep the reference model fixed throughout training. We evaluate the final generator on held-out prompts using Gradescope chrF.

5.3.3 Parphrase detection improvement

Table 2 summarizes the main hyperparameter configurations used in our paraphrase detection experiments, and Table 3 reports the SMART regularization parameters.

For optimization, we adopt a simple learning-rate schedule. Training typically begins with a relatively larger learning rate to allow faster progress in the early stages. When development performance stops improving, we reduce the learning rate to enable finer updates near convergence. In practice, we observe that continuing to increase the number of epochs does not always lead to better results, and excessive training can sometimes degrade performance due to overfitting. Therefore, training is usually stopped once improvements on the development set become marginal.

The batch size is primarily determined by GPU memory constraints. Larger models require smaller batch sizes to avoid out-of-memory errors. As shown in Table 2, GPT-2 Small can be trained with a batch size of 32, while GPT-2 Medium and GPT-2 Large require smaller batch sizes due to their larger memory footprint.

Model Information				
Experiment	Model	Strategy	Trainable Params	
Exp1	GPT-2 Small	-	110M	
Exp2	GPT-2 Small	LoRA	0.29M	
Exp3	GPT-2 Small	SMART	110M	
Exp4	GPT-2 Medium	LoRA	0.78M	
Exp5	GPT-2 Large	LoRA	1.47M	

Training Hyperparameters				
Experiment	Batch Size	Initial LR	Later LR	Epochs
Exp1	32	2×10^{-5}	-	8
Exp2	32	5×10^{-5}	2×10^{-5}	6
Exp3	16	1×10^{-4}	$5 \times 10^{-5}, 2 \times 10^{-5}$	8
Exp4	8	1×10^{-4}	1×10^{-5}	12
Exp5	4	1×10^{-4}	5×10^{-5}	8

Table 2: Hyperparameter configurations for paraphrase detection experiments.

For SMART regularization, we use the hyperparameters shown in Table 3. The perturbation magnitude ϵ_{smart} and noise variance are kept small to ensure that the adversarial perturbations remain close to the original embedding space, while the step size controls the strength of the adversarial update. The

coefficient λ_{smart} balances the task loss and the smoothness regularization term. In our experiments, these values provide stable training and improve robustness without noticeably slowing convergence.

λ_{smart}	ϵ_{smart}	Step size	Noise variance	Update steps
0.02	1×10^{-5}	1×10^{-3}	1×10^{-5}	1

Table 3: SMART regularization hyperparameters used in paraphrase detection experiments.

5.4 Results

We discuss the effects of LoRA and SMART regularization on paraphrase accuracy, and the effects of DPO and decoding choices on sonnet chrF and the role of pair construction and decoding in Section 6.

Task / Setting	Metric	Score
SST last-linear-layer (ours)	Dev acc	0.47
SST full-model (ours)	Dev acc	0.512
CFIMDB last-linear-layer (ours)	Dev acc	0.87
CFIMDB full-model (ours)	Dev acc	0.98
Sonnet Dev (ours)	Dev chrF	42.34
Sonnet Test (ours)	Test chrF	41.864
Paraphrase Dev (ours)	Dev acc	0.902
Paraphrase Test (ours)	Test acc	0.883
SST last-linear-layer (STAFF baseline)	Dev acc	0.462
SST full-model (STAFF baseline)	Dev acc	0.513
CFIMDB last-linear-layer (STAFF baseline)	Dev acc	0.861
CFIMDB full-model (STAFF baseline)	Dev acc	0.976
Sonnet Dev (STAFF baseline)	Dev chrF	41.414
Sonnet Test (STAFF baseline)	Test chrF	41.208
Paraphrase Dev (STAFF baseline)	Dev acc	0.882
Paraphrase Test (STAFF baseline)	Test acc	0.856

Table 4: Final results after hyperparameter tuning vs. staff baselines. Sonnet scores were improved mainly through tuning epochs and decoding parameters.

6 Analysis

6.1 Sonnet generation

Our sonnet experiments suggest that generation quality on this task depends not only on fitting the training data, but also on how well the model preserves structure during decoding. Compared with standard likelihood-based training alone, the preference-based objective appears to help the model distinguish between structurally better and worse continuations. In our setting, the negative examples were created by corrupting the true continuation, so the main signal learned by the model was not topic relevance, but structural consistency. This is likely one reason why the method improved chrF: the preferred continuations remain close to the original sonnet in line order and local phrasing, while the corrupted ones break this structure.

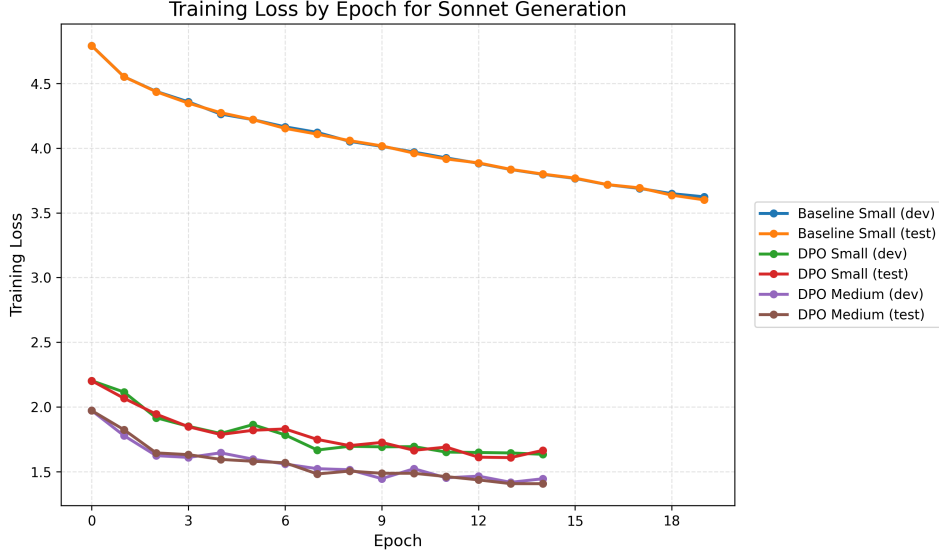


Figure 1: Training loss curves for multiple sonnet-generation runs, including baseline GPT-2 Small and DPO-based models on dev/test splits. All models show rapid loss reduction in early epochs, followed by slower improvement later in training. The DPO-based runs generally converge to lower loss than the baseline runs, but lower training loss does not always correspond directly to higher chrF on held-out prompts.

Figure 1 compares training loss across several sonnet-generation runs. Across all settings, the loss decreases quickly in the early epochs and then flattens, showing that the sonnet dataset is learned relatively fast. At the same time, the curves also show that lower training loss alone is not enough to identify the best model. Some runs achieved lower loss without producing the best chrF on held-out prompts. This confirms that, for sonnet generation, final performance depends not only on optimization during training but also on the decoding strategy used at inference time.

We also found that chrF was highly sensitive to decoding choices. This makes sense because chrF rewards local overlap with the reference text, so even a reasonably fluent continuation can score poorly if its wording drifts too far from the gold continuation. In practice, higher temperature helped the model avoid overly rigid and repetitive outputs, while a smaller top- p constrained decoding enough to preserve more stable local structure. The best results came from balancing these two effects: enough randomness to avoid dull or repetitive text, but enough restriction to keep the continuation closer to the style and phrasing of the reference.

At the same time, our results highlight a limitation of this setup. The automatically constructed negative examples mainly capture structural corruption, such as line reordering or deletion, but they do not fully represent all real generation errors. For example, sampled outputs may still be grammatically awkward, semantically off-topic, or excessively repetitive in ways that are different from our corrupted negatives. This means the learned preference signal is useful but still narrow. Overall, the sonnet task appears to benefit from combining three factors: a preference-based training signal, a stabilizing language-modeling objective, and careful decoding control at inference time. The comparison of loss curves also suggests that different training settings can follow similar optimization trends while still producing different held-out chrF scores.

6.2 Paraphrase detection

As illustrated in Figure 2, we first applied LoRA to GPT-2 Small and observed that the performance remained comparable to the baseline while significantly reducing the number of trainable parameters, achieving a development accuracy of approximately 0.843. In this setting, the number of trainable parameters decreases from roughly 110M in the full-model baseline to only about 0.29M when using LoRA.

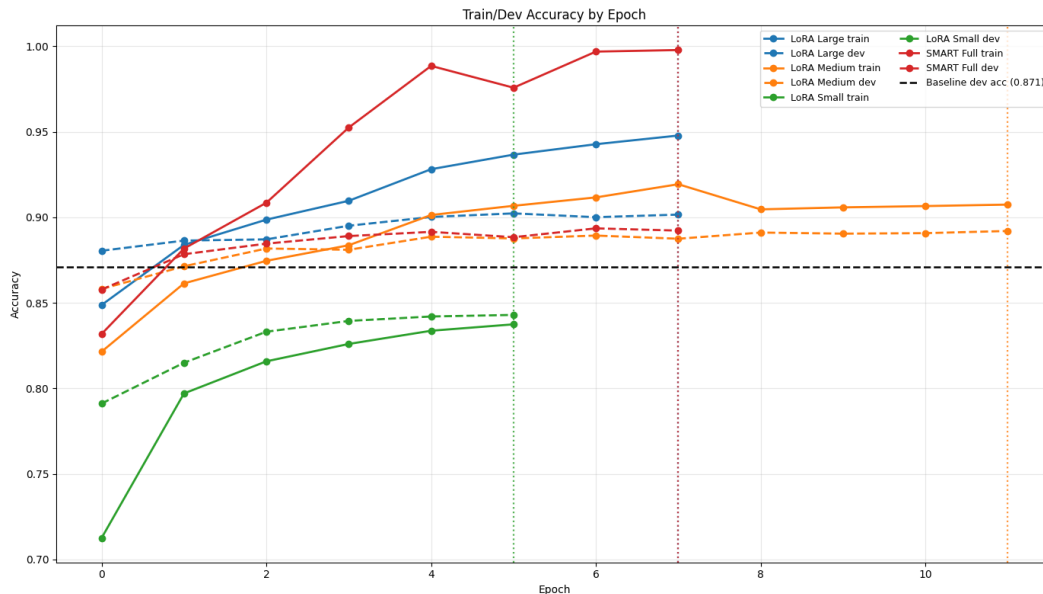


Figure 2: Training and validation accuracy across epochs for different model configurations (paraphrase detection task). The dashed horizontal line indicates the baseline dev accuracy, while vertical dotted lines mark the epoch at which each model achieves its best validation performance. The number of training epochs also varies across configurations, as it is determined by both the computational time and the observed convergence behavior.

We then scaled the base model while keeping the LoRA architecture unchanged. With **GPT-2 Medium + LoRA**, the development accuracy improved to **0.892**, and with **GPT-2 Large + LoRA**, it further increased to **0.902**. Despite the larger backbone models, the number of trainable parameters remained relatively small (about 0.78M for GPT-2 Medium and 1.47M for GPT-2 Large), which is still significantly smaller than full-model fine-tuning. This allows us to benefit from larger models while keeping training computationally efficient.

Finally, we compared SMART regularization with the LoRA-based approaches. Our results show that GPT-2 Small with SMART achieves a development accuracy of **0.8922**, which is comparable to GPT-2 Medium with LoRA (dev accuracy **0.8919**). This suggests that adversarial regularization can improve robustness and provide performance gains similar to scaling the base model.

7 Conclusion

In this project, we implemented the main components of GPT-2 from the CS224N scaffold and validated our implementation across the default benchmark tasks. Our sentiment results served as an important sanity check and showed that the model and optimization pipeline were working as expected. Building on this foundation, we then studied two downstream tasks in more detail: paraphrase detection and sonnet generation.

For paraphrase detection, we found that LoRA provided an effective and efficient way to improve performance while greatly reducing the number of trainable parameters. Scaling from GPT-2 Small to Medium and Large further improved development accuracy, showing that larger pretrained backbones remain useful even in a LoRA setting. We also found that SMART regularization substantially strengthened the full-model GPT-2 Small baseline, reaching performance comparable to GPT-2 Medium + LoRA.

For sonnet generation, our experiments showed that performance depended on both the training objective and the decoding strategy. Using DPO with automatically constructed preference pairs helped the model learn structural preferences beyond standard next-token prediction, while the added continuation-only cross-entropy term improved training stability. At the same time, decoding choices

had a strong effect on final chrF, and careful tuning of temperature, top- p , learning rate, and number of epochs was necessary to obtain the best results. Our best configuration achieved 42.34 chrF on the development set and 41.864 chrF on the test set.

Overall, task-specific optimizations can enable the GPT-2 model to achieve comparable performance across different applications.

Team contributions (Required for multi-person team)

Kayla Li and Yaojing Huang He split the work equally across implementation, experiments, and writing. Both authors helped debug the codebase, tune hyperparameters/decoding settings, and iteratively refine the report based on experimental outcomes.

Kayla focused on the GPT-2 model implementation and sonnet generation pipeline. This included completing key TODO components needed for end-to-end runs, running sentiment and sonnet training on local hardware, and implementing the DPO-based sonnet extension (pair construction, reference-model freezing, and the mixed CE+DPO objective).

Yaojing focused on implementing the AdamW optimizer and improving the paraphrase detection pipeline. This included implementing and validating AdamW, supporting training and debugging across tasks, and developing the paraphrase detection baseline and improvements using LoRA and SMART regularization, as well as running experiments and recording leaderboard results.

References

- [1] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. *OpenAI Blog*, 2019.
- [2] Ashish Vaswani, Noam Shazeer, Niki Parmar, et al. Attention is all you need. In *NeurIPS*, 2017.
- [3] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.
- [4] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 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, 2021.
- [6] Haoming Jiang, Pengcheng He, Weizhu Chen, Xiaodong Liu, Jianfeng Gao, and Tuo Zhao. Smart: Robust and efficient fine-tuning for pre-trained natural language models through principled regularized optimization. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, page 2177–2190. Association for Computational Linguistics, 2020.
- [7] 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. *arXiv preprint arXiv:2305.18290*, 2024.
- [8] 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.