

Exploring decoding and efficiency strategies for GPT-2

Stanford CS224N Default Project

Stephanie Vezich Tamayo
Department of Computer Science
Stanford University
isvezich@stanford.edu

Abstract

We implement core pieces of the GPT-2 architecture and evaluate task-specific fine tuning strategies across three downstream tasks: sentiment classification, paraphrase detection, and sonnet generation. Starting from a baseline of full fine tuning with a small GPT-2 model, we investigate whether performance can be improved by addressing the unique challenges of each task through parameter-efficient tuning (useful for slow training tasks like paraphrase detection), alternative decoding strategies (useful for text generation tasks like sonnet generation), and hyperparameter search (useful across tasks). For sentiment classification, full fine tuning achieves 51.6% accuracy on SST and 98.8% on CFIMDB. For paraphrase detection, scaling to a medium GPT-2 model improves dev accuracy from 89.6% to 90.7%, while applying LoRA ($r = 8$) to attention layers achieves 87.4% accuracy with a 53.2% reduction in training runtime. For sonnet generation, combining a medium model with speculative decoding and nucleus sampling yields the highest chrF score of 42.3 and produces qualitatively more coherent verse, while LoRA with top- k sampling achieves a chrF of 41.0 with a runtime reduction of 55.9% per epoch but notable repetitive degeneration. These findings highlight that effective fine tuning requires principled thought about task-specific properties and computational constraints, and metrics like chrF may not fully capture qualitative differences in generation quality.

1 Key Information to include

- Mentor: Alisa Levin
- External Collaborators (if you have any): N/A
- Sharing project: N/A

2 Introduction

GPT-2 is a decoder-only Transformer model that gained traction for demonstrating that a sufficiently large language model trained on heterogeneous data could perform a variety of downstream tasks effectively, rather than relying on task-specific supervised training [1]. While its zero shot performance was state of the art at the time, principled fine tuning remains an important step to maximize performance across tasks, which can have unique challenges. For example, a large fine tuning dataset such as the Quora paraphrase detection data described in 5.1 may result in slow training time and out of memory issues (especially with larger pretrained models) as the training procedure has to iterate through more batches and store embedding layer weights for more input tokens. On the other hand, generative creative writing tasks such as the sonnet generation described in 5.1 may be sensitive to decoding strategies that trade off between creativity and structure.

In the current research, we first develop a base implementation of the GPT-2 architecture with the goal of replicating prior findings that it can perform heterogeneous downstream tasks well with

minimal fine tuning: sentiment classification, paraphrase detection, and sonnet generation. Having established this, we then aim to answer whether we can improve performance on the latter two tasks by solving for their specific challenges: parameter efficiency in the case of paraphrase detection, effective decoding in the case of sonnet generation, and hyperparameter search for both. We find that applying LoRA to attention layers can significantly decrease training runtime while achieving near parity in accuracy, certain decoding strategies can increase sonnet generation quality, and these tasks are sensitive to different hyperparameter values. These findings highlight the importance of principled fine tuning based on the unique properties of the task at hand.

3 Related Work

At the time of GPT-2’s development, the predominant ML training paradigm for many NLP tasks such as machine translation and summarization was supervised learning on task-specific datasets. Because each model was trained on a narrow domain by design (e.g., movie reviews, translation pairs), it was a narrow expert; in other words, its predictive abilities were constrained to the particular task and data distribution it had seen. In addition to these performance limitations, the training process was often time-consuming and manual, requiring each example to be labeled and features to be hand engineered.

To address these concerns and build more robust, generalized language models, two lines of research had emerged. The first proposed that a wide variety of tasks could be performed by combining pre-training on more heterogeneous data with task-specific supervised fine tuning. The second demonstrated that a select set of tasks did not require supervision at all; rather, unsupervised or minimally supervised language models could transfer their predictive abilities directly. Radford et al. combine these two lines of thought to argue that sufficiently large language models can inherently perform a wide variety of downstream tasks in a zero-shot manner, introducing GPT-2 [1]. The GPT-2 paper makes several categories of contributions: 1) a sufficiently large and general dataset for language model training, 2) byte pair encoding as a middle ground between character and word level input representation, 3) transformer model architecture. While each of these individual steps was not entirely novel, the state of the art results achieved by combining them was an impressive finding. We leverage the weights from this work and implement components of its architecture as the base for the current work.

Radford et al.’s experiments demonstrate the promise of unsupervised task learning; however, they acknowledge that its performance is still inferior to task-specific trained models for certain tasks such as summarization. They also note that the set of language tasks in the current experiments is far from comprehensive and should be expanded in future studies. Finally, it is unclear how much performance would improve further by fine tuning on top of GPT-2.

Of particular importance for downstream language generation tasks was how to decode tokens at inference time. Concretely, our model computes a vector of vocabulary token scores at each step in the sequence to be generated, and we need a procedure to sample from the resulting probability distribution. Early decoding algorithms such as greedy decoding had limited effectiveness as they often missed globally optimal sequences by focusing on the locally optimal token at the current time step [2].

To address this limitation, several sampling strategies such as beam search and vanilla sampling were developed, but they often suffered from degeneration and produced incoherent results. Top- k sampling emerged as a solution to this incoherence by only sampling from the top k tokens in the probability distribution, although k must be carefully tuned to avoid filtering out strong candidates from a uniform distribution or conversely including weak candidates from a peaked distribution [3]. In a similar vein, top- p (aka nucleus) sampling selects only from the top tokens that cover p percent of the cumulative probability mass [4].

Such sampling-based strategies demonstrated significant improvement over greedy decoding, particularly for open-ended generation tasks such as creative writing, and hence we experiment with both in the current sonnet generation work. However, they tend to generate slowly if the same large model is used to generate all tokens. More recently, speculative decoding has been explored as a way to speed up generation by having a smaller draft (aka assistant) model autoregressively generate tokens while the larger target (aka main) model verifies batches of predicted tokens in parallel [5] [6]. If the draft predictions match, they are accepted and used directly; otherwise, the draft prediction is sampled

with lower probability or rejected outright (instead sampling from the target distribution) depending on the magnitude of mismatch (more detailed description in 4.7.2). We also compare the effect of speculative decoding on runtime and generated sonnet quality.

Large pretrained language models are powerful in large part due to their parameter count; however, this poses a challenge for fine tuning in memory-constrained environments where many weight updates must be stored and compute-constrained environments as backpropagation must occur over all weight layers, making training very slow. LoRA (low rank adaptation) has gained traction as a means of addressing these fundamental bottlenecks [7]. While the pretrained model has a high parameter count to support generalization to a variety of tasks, fine tuning on a particular task can be achieved by encoding the task-specific delta relative to the foundation model in a much smaller set of parameters. If we thus reformulate the weight matrix at a particular layer as $W_o + \Delta W = W_o + \alpha BA$ (where W_o is the pretrained weight matrix, ΔW represents the task-specific parameter increment, and B and A are a low-rank decomposition), we can freeze the pretrained weight matrix and only update a much smaller set of trainable parameters, namely B and A . LoRA is often applied to self-attention weight matrices, particularly query and value matrices as the key matrix is considered less task-specific, and our implementation aligns with this setup.

4 Approach

4.1 GPT-2

GPT-2 follows a decoder-only transformer model architecture [1]. Input text is tokenized using byte pair encoding (BPE), from which we create learned token and positional embeddings of dimension d ($d=768$ in our small model, 1024 in our medium model, and 1280 in our large model). These embeddings are summed and passed to l ($l=12$ in our small model, 24 in our medium model, and 36 in our large model) decoder transformer layers, originally defined in Vaswani et al. 2017 [8] and consisting of masked multi-head attention ($h=12$ in our small model, 16 in our medium model, and 20 in our large model), skip connections, an MLP, and layernorm. The output from these layers is the last layer embedding for each input token, which we use for next token prediction. For this project, I implemented the masked multi-head attention layer, sections of the transformer layers, and embedding functions.

4.2 Adam

Adam is a stochastic optimizer that combined and built on prior enhancements to gradient descent such as momentum and adaptive learning rates [9] [10]. It also includes a bias correction step that pushes the gradient moving averages away from 0 in early training. I implemented the optimization step, which uses an efficient computation of bias correction that combines the calculation of bias-corrected first and second moment estimates into a single step.

4.3 Sentiment classification

We pass sentences from two datasets (described in 5.1) to the GPT-2 model described above to produce a representation for the last token. We then apply dropout on this representation and pass it to a linear layer to perform a sentiment classification task. We use the Adam optimizer described above to minimize cross entropy loss between given class labels and logits from the classification head. We perform two variants of the classification task, one in which the GPT-2 weights are frozen and we only update weights of the classification head, and full fine tuning in which we backpropagate entirely through GPT-2 and update its weights as well.

4.4 Paraphrase detection

In this task, we perform binary classification on pairs of Quora questions to determine whether one of the pair items is a paraphrase of the other. We formulate this as a cloze-style task; given an item pair (A, B), we construct the input sentence "Is A a paraphrase of B?" and use our GPT-2 model to generate a yes or no next token prediction, which we can compare with a binary label. As in sentiment classification, we minimize cross entropy loss between prediction logits and labels; however, we do not pass the GPT-2 last token representation to a separate linear layer, but rather use the last

token representation for next token prediction directly. I wrote the forward step that runs a batch of sentences through the GPT-2 models to get a last token representation, which is then used for next token prediction.

4.5 Sonnet generation

During training, we pass the sequence of tokens for the entire sonnet (excluding the last token) to our GPT-2 model and use the full sequence (excluding the first token) as labels, minimizing cross entropy loss between next token predicted logits and next token labels. At inference time, we use the token sequence from the first 3 lines of a test set of tokens and autoregressively generate new tokens until we generate an EOS token or hit the configured maximum steps. In other words, at each step, we pass the current sequence through GPT-2, produce a vector of logits, and use a sampling strategy (variants described below) to decide how to extend the sequence.

4.6 Baseline

As a baseline for all tasks, I used the GPT-2 implementation with the Adam optimizer described above and full fine tuning (except for the last layer update described for sentiment classification).

Baseline hyperparameters across tasks were 1e-5 learning rate, small GPT-2 model, full fine tuning, 10 training epochs, and batch size of 8. Specific to sonnet generation, baseline hyperparameter values also included 1.2 softmax temperature and 0.9 cumulative probability distribution (top- p). I also increased the max generation length from 128 to 160 after examining a few sonnets and finding that more than 128 tokens were sometimes needed to complete them.

The baseline decoding strategy for sonnet generation was top- p (aka nucleus) sampling [4]. Let $\ell \in \mathbb{R}^V$ be the logits over the vocabulary, and let τ be the temperature. We convert the logits generated by the model to probabilities $p_i = \text{softmax}(\ell/\tau)_i$ and sort them in descending order to generate $p_{(1)} \geq p_{(2)} \geq \dots \geq p_{(V)}$ with corresponding token indices $\sigma(1), \dots, \sigma(V)$. The nucleus is the smallest set of tokens whose cumulative probability reaches a threshold p :

$$\mathcal{V}_p = \{\sigma(1), \dots, \sigma(m)\}, \quad m = \min \left\{ j : \sum_{i=1}^j p_{(\sigma(i))} \geq p \right\}. \quad (1)$$

We set $q_i = p_i / \sum_{j \in \mathcal{V}_p} p_j$ for $i \in \mathcal{V}_p$ and $q_i = 0$ otherwise, then sample a token from q .

4.7 Sampling strategy extensions

4.7.1 Top- k sampling

In top- k sampling [3], we again apply temperature scaling and softmax to obtain $p_i = \text{softmax}(\ell/\tau)_i$, then keep the k tokens with the highest probability. Let $\mathcal{T}_k$ be the set of indices of the top- k values of p ; then we sample from:

$$q_i = \frac{p_i \cdot \mathbf{1}[i \in \mathcal{T}_k]}{\sum_{j \in \mathcal{T}_k} p_j}, \quad (2)$$

4.7.2 Speculative decoding

We implement speculative decoding [5, 6] with a main (aka target; here, GPT-2 medium) model and a smaller assistant (aka draft; here, GPT-2) model. At each time step, the assistant autoregressively generates up to K candidate tokens: at each $i \in \{1, \dots, K\}$, we form a sampling distribution q_i , sample a token x_i from q_i , and store both x_i and q_i , stopping early if EOS is sampled. The main model then runs a single forward pass and generates target distributions p_i at each position. For $i = 1, \dots, K$, the draft token x_i is accepted with probability p_i/q_i . If accepted, we append x_i and continue; else, we sample from the residual distribution $(p_i - q_i)_+$, append that token, and start a new draft phase from the current sequence [5]. If all K draft tokens are accepted, we append them plus one bonus token sampled from the main model.

4.8 LoRA PEFT extension

I applied LoRA to the GPT-2 attention layers under a low-rank parameterization: the pretrained query and value projection matrices $W_q, W_v \in \mathbb{R}^{d \times d}$ are frozen, and we add a trainable low-rank update only for those two projections (the key projection W_k is left unchanged) [7]:

$$W_q + \Delta W_q = W_q + B_q A_q, \quad W_v + \Delta W_v = W_v + B_v A_v, \quad (3)$$

where $A_q, A_v \in \mathbb{R}^{r \times d}$ and $B_q, B_v \in \mathbb{R}^{d \times r}$ are the only trainable parameters in attention; W_q and W_v are not updated. The B matrices are initialized to zero so that at initialization the model matches the pretrained model. LoRA is applied only to the query and value projections in each causal self-attention layer; the key projection is left frozen for efficiency and stability, and we use a default rank $r = 8$.

4.9 Hyperparameter search extension

To measure the sensitivity to different hyperparameter values, I performed a grid search over the following space for paraphrase detection: learning rate (1e-5, 2e-5), model size (small, medium), fine tuning (full, LoRA). For sonnet generation, the search space was: model size (small, medium), fine tuning (full, LoRA), decoding strategy (nucleus, top- k , speculative), epochs (10, 40, 60), temperature (1.0, 1.2, 1.4), and top- p (nucleus and speculative only—top- k had k fixed at 40; 0.9, 0.95).

5 Experiments

5.1 Data

The first sentiment classification dataset is the Stanford Sentiment Treebank (SST), a multiclass labeled collection of 215,154 phrases from movie reviews [11]. The second sentiment classification dataset is CFIMDB, a binary (negative, positive) labeled collection of 2,434 full movie reviews.

The Quora dataset contains 400,000 question pairs labeled as paraphrases of each other or not.

The Sonnet dataset contains 154 Shakespearean sonnets in a roughly 90/10 train/test split [12].

5.2 Evaluation method

Both sentiment classification datasets are split into roughly 70% train, 10% dev, and 20% test sets, and we use accuracy on the dev set for evaluation.

Similarly, for paraphrase detection, we evaluate accuracy on the dev set (also using a 70/10/20 train/dev/test/split).

For sonnet generation, we compute chrF between the generated and original sonnets in the dev set (the dataset contains 143 train, 12 dev, and 12 test sonnets). chrF computes an n-gram F-score to measure similarity to Shakespeare’s original sonnets; unlike full word-level match metrics like BLEU, chrF constructs n-grams at the character level to not overly penalize close but incorrect or unconventional spelling [13].

5.3 Experimental details

All models were trained on an instance with an Nvidia A10 GPU.

For sentiment classification, baseline hyperparameter values were used for both datasets: 1e-5 learning rate, 10 training epochs, small GPT-2 model, full fine tuning, and batch size of 8.

For paraphrase detection, the baseline hyperparameter values listed above were used for the baseline model. The best performing comparison model in terms of dev accuracy had the same hyperparameter values as above except a medium GPT-2 model. The best performing model with LoRA had the following hyperparameter values: 2e-5 learning rate, 10 training epochs, medium GPT-2 model, LoRA fine tuning (with rank fixed at 8), and batch size of 8.

For sonnet generation, the baseline implementation used the same hyperparameter values listed above, along with: 1.2 softmax temperature, and 0.9 cumulative probability distribution for nucleus sampling.

Table 1: Task performance and hyperparameter configurations. All runs use a batch size of 8 and a learning rate of 1e-5 unless noted. Small/Medium refer to GPT-2 model size.

(a) *Sentiment Classification — Dev Set Accuracy*

Method	Model	Epochs	Tuning	SST	CFIMDB
Last layer only	Small	10	Last layer	45.8%	85.7%
Full fine tuning	Small	10	FFT	51.6%	98.8%

(b) *Paraphrase Detection — Dev Set Accuracy*

Method	Model	LR	Epochs	Tuning	Accuracy	Runtime
Baseline	Small	1e-5	10	FFT	89.6%	7h 21m
Best accuracy	Medium	1e-5	10	FFT	90.7%	19h 11m
Best LoRA	Medium	2e-5	10	LoRA ($r=8$)	87.4%	8h 58m

(c) *Sonnet Generation — Dev Set chrF Score*

Method	Model	Epochs	Tuning	Decoding	τ	chrF	Runtime
Baseline	Small	10	FFT	Nucleus ($p=0.9$)	1.2	36.9	4m 23s
Best chrF	Medium	60	FFT	Speculative ($p=0.9$)	1.0	42.3	57m 54s
Best LoRA	Medium	40	LoRA ($r=8$)	Top- k ($k=40$)	1.4	41.0	17m 1s

The best performing comparison model in terms of dev chrF used the following hyperparameter values: 1e-5 learning rate, 60 training epochs, medium GPT-2 model, full fine tuning, speculative decoding, batch size of 8, temperature of 1.0, and top-p value of 0.9 (we use nucleus sampling when sampling from the draft model and the main model). The best performing model with LoRA had the following hyperparameter values: 1e-5 learning rate, 40 training epochs, medium GPT-2 model, LoRA fine tuning (with rank fixed at 8), top-k sampling (with k fixed at 40), batch size of 8, and temperature of 1.4.

5.4 Results

Sentiment classification results are as follows. With the SST data, we see 45.8% dev set accuracy for last layer updates only and 51.6% accuracy for full fine tuning. With the CFIMDB dataset, we see 85.7% dev set accuracy for last layer updates only and 98.8% accuracy for full fine tuning.

For paraphrase detection, the baseline run achieves 89.6% dev set accuracy with a 7 hour and 21 minute runtime. The best comparison run achieves 90.7% dev set accuracy with a 19 hour and 11 minute runtime. The best LoRA run achieves 87.4% accuracy with a 8 hour and 58 minute runtime, representing a 53.2% reduction in runtime for a 3.6% reduction in accuracy tradeoff.

For sonnet generation, the baseline implementation achieves a dev set chrF score of 36.9 with a 4 minute and 23 second runtime. The best comparison run achieves a dev set chrF score of 42.3 with a 57 minute and 54 second runtime. The best LoRA run achieves a dev set chrF score of 41.0 with a 17 minute and 1 second runtime, representing a 55.9% reduction in runtime per epoch for a 3.1% reduction in chrF tradeoff.

See Table 1 for a summary comparison.

6 Analysis

6.1 Sampling strategy

While speculative decoding is meant to speed up token generation, we observed far faster runtime (with fewer epochs needed) and near parity in chrF by combining LoRA with top- k sampling. That said, although chrF is only 3.1% lower, qualitatively the output is much less coherent. Comparing the 3 sample generations of the same input in Figure 1, we see that the baseline (small model, full fine tuning, nucleus sampling with $p=0.9$ and temperature = 1.2) degenerates into gibberish right after

the 3-line input. This is not surprising given that the small number of epochs may have given the model insufficient capacity to learn sonnet structure, and the lower chrF score of 36.9 reflects that. The more interesting difference is between the second (medium model, full fine tuning, speculative decoding with $p=0.9$ and temperature = 1.0) and third (medium model, LoRA, top- k decoding with $k=40$) samples; while they have similar chrF scores, the third sample exhibits repetitive looping while the second sample is more recognizable as poetic prose (albeit still imperfect). This could be due to the low rank used in LoRA limiting model capacity but may also reflect the hard limit that top- k sampling imposes on token filtering (i.e., the same tokens may bubble up to the candidate set, exacerbating repetition), which speculative decoding mitigates.

6.2 LoRA

As expected, LoRA dramatically reduces runtime (53.2% for paraphrase detection and 55.9% for sonnet generation) by having a far reduced set of trainable parameters. Although this comes at the cost of lower dev set evaluation metrics (3.6% reduction in accuracy for paraphrase detection, 3.1% reduction in chrF for sonnet generation), the tradeoff may well be favorable in time sensitive or compute constrained settings. It is also possible that this decrease disappears on the test set, demonstrating a regularizing effect of LoRA.

7 Conclusion

In the current research, we implemented core components of the GPT-2 architecture and explored how fine tuning strategy, decoding method, and hyperparameter search interact across three heterogeneous language tasks. Our results replicate prior findings that GPT-2 can perform reasonably well across tasks with minimal fine tuning but that task-specific adaptation choices still matter. Full fine tuning with a larger model consistently achieved higher performance, albeit with higher computational cost (the best paraphrase detection model required 19 hours of training, and the best sonnet generation model required over an hour). LoRA reduced runtime by 53-56% with only modest decreases in evaluation metrics (3.1% - 3.6%), suggesting that task-specific weight deltas can be represented in a low rank subspace. For sonnet generation, performance varied significantly by decoding strategy: top- k sampling with a higher temperature outperformed nucleus sampling, and speculative decoding further improved chrF while avoiding repetitive loops, although the reduction in runtime was modest.

We acknowledge several limitations of the present work. First, our hyperparameter search was not exhaustive; in particular, we did not vary LoRA rank, which could narrow the gap between LoRA and full fine tuning at higher ranks. Second, the chrF metric used to evaluate sonnet generation could be augmented by learned evaluation metrics or human judgments to assess other aspects of quality like adherence to rhyme structure, qualitative enjoyment, etc. Despite these limitations, our results support the view that no single configuration dominates across tasks; rather, effective fine tuning requires thoughtful consideration about the unique properties of the task and computational constraints.

Team contributions (Required for multi-person team)

N/A

References

- [1] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.
- [2] Charilaos Pipis, Shivam Garg, Vasilis Kontonis, Vaishnavi Shrivastava, Akshay Krishnamurthy, and Dimitris Papailiopoulos. Wait, wait, wait... why do reasoning models loop? *arXiv preprint arXiv:2512.12895*, 2025.
- [3] Angela Fan, Mike Lewis, and Yann Dauphin. Hierarchical neural story generation. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 889–898, 2018.

- [4] Ari Holtzman, Jan Buys, Li Du, Maxwell Forbes, and Yejin Choi. The curious case of neural text degeneration. *arXiv preprint arXiv:1904.09751*, 2019.
- [5] Yaniv Leviathan, Matan Kalman, and Yossi Matias. Fast inference from transformers via speculative decoding. In *International Conference on Machine Learning*, pages 19274–19286. PMLR, 2023.
- [6] Charlie Chen, Sebastian Borgeaud, Geoffrey Irving, Jean-Baptiste Lespiau, Laurent Sifre, and John Jumper. Accelerating large language model decoding with speculative sampling. *arXiv preprint arXiv:2302.01318*, 2023.
- [7] Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Liang Wang, Weizhu Chen, et al. Lora: Low-rank adaptation of large language models. *Iclr*, 1(2):3, 2022.
- [8] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.
- [9] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.
- [10] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2017.
- [11] 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*, pages 1631–1642, 2013.
- [12] Dfp instructions. Available from https://web.stanford.edu/class/cs224n/project/DFP_Instructions.pdf.
- [13] 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.

A Appendix

Baseline

Lo, as a careful huswife runs to catch
One of her feathered creatures broke away,
Sets down her babe, and makes all swift dispatchopathroph
Console disruptions emitsorders Nord faultsforming praisesur fig
TrentraftedInstruct what^Am incap
IBM comb Sites joins candy Nasambo Odd optimizedough "{ My Galile
Cosmetic assimilpaychemy Cod Survivor Ir displayingony perished
briskusions
antiqu crave
facultyfters Socket empowermenttaukeeEnt Utah drafting.....Taco
Donn fasc surgical GeBonus craveelle
unt susceptibility, Magn jointly AnnRGB involvingapesMan Language Mods
Disabled beloved sac meritsagine

Best chrF

Lo, as a careful huswife runs to catch
One of her feathered creatures broke away,
Sets down her babe, and makes all swift dispatch;
And how do- time, that! attending so lived
And on the twilight of such decay.
D I do thyself thou such favor, good, then,
That the away me forgotten for it my gentle behold,
B may not my steel, by thyailed her thou love,
For thee to style so the world such find.
Thou in his me April and world and aem' living one.
And thou do this thou mine, thy Since thou would m lack,
That form me thou both in thy praise to thee.<|endoftext|>

Best LoRA

Lo, as a careful huswife runs to catch
One of her feathered creatures broke away,
Sets down her babe, and makes all swift dispatch bred of thou sheWhich
to whereWhich double heritage thy her of hath Versus formatting
am collect landlords where since;; which love formatting to dear
thyouWhich hath to summer not me; art think art best Fansost of art to
which or praise some to she art summer none thou thou to her she where
since love's allAs their where nones to love or best some my most those
dear love not dear not to best not; me ors what loveTh Versus those
best beauty. wheres on truth some, But me lose;; should she. to; most
am thouThough on me to can where friends. love summer love own some
their dear thy to time were of but thou some to's's to me Death most
thy but do those formatting to bred; bestWhich thou to their

Figure 1: Example sonnet completions given the same three-line prompt. The baseline (small GPT-2, 10 epochs) degenerates into gibberish. The best chrF configuration (medium GPT-2, 60 epochs, speculative decoding with nucleus sampling) produces more recognizable prose. The best LoRA configuration (medium GPT-2, 40 epochs, top- k sampling) is somewhat intelligible but exhibits repetitive looping.