

Build GPT-2

Stanford CS224N Default Project

Lucia Losada

Department of Computer Science
Stanford University
lulosada@stanford.edu

Nicole Cortes

Department of Computer Science
Stanford University
nicortes@stanford.edu

Abstract

Adapting large pre-trained language models to downstream tasks efficiently and accurately is a central challenge in modern NLP. We implement GPT-2 from scratch in PyTorch and fine-tune it on three tasks: sentiment classification, cloze-style paraphrase detection, and sonnet generation. We then extend the pipeline with Low-Rank Adaptation (LoRA), integrating trainable low-rank matrices directly into GPT-2 attention projections without external libraries. A rank sweep over $r \in \{1, 2, 4, 8, 16\}$ on sonnet generation reveals a clear quality-rank trend: full fine-tuning achieves chrF = 32.5, while LoRA at $r = 16$ reaches chrF = 28.9 using under 0.5% of parameters. Extending LoRA to paraphrase detection yields 79.5% dev accuracy (at $r=16$) versus 88.5% for full fine-tuning, indicating greater task-dependent sensitivity to the low-rank constraint. These results suggest that parameter-efficient fine-tuning provides a strong balance between performance and efficiency, although the size of the performance gap varies across different tasks and may require task-specific tuning.

1 Key Information

- TA mentor: Alisa Levin
- External collaborators / External mentor / Sharing project: No

2 Introduction

Pre-trained transformer language models have become a standard foundation for NLP, but effectively adapting them to downstream tasks requires deep understanding of their internals. In this project, we implement GPT-2 [1] from scratch in PyTorch and fine-tune it on three structurally distinct tasks: sentiment classification, cloze-style paraphrase detection, and open-ended sonnet generation. Each task focuses on a different capability of the model, making the setup a useful multi-task evaluation.

Full fine-tuning updates all 125M GPT-2 parameters, which is memory-intensive and risks overwriting useful pretrained representations. We address this with Low-Rank Adaptation (LoRA) [2], implemented directly in our codebase without external PEFT libraries. LoRA freezes the pretrained weights and injects trainable low-rank matrices into attention projections, which reduces the number of trainable parameters. We apply LoRA to sonnet generation via a rank sweep over $r \in \{1, 2, 4, 8, 16\}$, evaluated with chrF [3], and extend the same integration to paraphrase detection. LoRA reaches chrF = 28.9 at $r = 16$ versus 32.5 for full fine-tuning while training under 0.5% of parameters. On paraphrase detection, full fine-tuning achieves 88.5% dev accuracy while LoRA peaks at 79.5% ($r = 16$) with only 0.5% of parameters trained.

Our main contributions are:

- A from-scratch GPT-2 implementation with causal self-attention, pre-norm residual blocks, weight-tied output projection, and a custom AdamW optimizer [4].

- Fine-tuning pipelines for sentiment classification (last-linear-layer and full-model), paraphrase detection, and sonnet generation with nucleus sampling [5].
- A LoRA rank sweep for sonnet generation and paraphrase detection, characterizing the quality–efficiency tradeoff of low-rank adaptation.

3 Related Work

GPT-2 [1] benefits strongly from large-scale pretraining, and it is typically adapted to downstream tasks through full fine-tuning. While this approach works well, updating all parameters is costly and makes it harder to maintain separate models for different tasks.

LoRA for parameter-efficient fine-tuning. LoRA [2] freezes pretrained weights and learns low-rank update matrices for selected linear layers, often reaching similar performance while training far fewer parameters. Our work follows this setup and evaluates LoRA behavior in the CS224N default project tasks.

Generation decoding. Open-ended generation quality is sensitive to decoding strategy. Holtzman et al. [5] show that standard decoding methods can lead to repetitive or low-quality text, which motivates sampling-based alternatives such as nucleus sampling. Fan et al. [6] also use top-k sampling for long-form generation, showing that limiting sampling to the top k candidates can help avoid poor choices from the low-probability tail. Temperature is another simple decoding setting that controls how spread out the next-token distribution is: higher temperatures usually produce more diverse outputs, while lower temperatures make generation more focused and predictable. Lee et al. [7] further show that temperature-based calibration can improve generation quality by reducing repetition. In our experiments, we also use top-k sampling and temperature as practical decoding choices when generating sonnets with LoRA.

4 Approach

We implement GPT-2 from scratch in PyTorch, building on the provided starter code (github.com/stanfordnlp/cs224n_gpt), which supplies the training loops, data loading, and evaluation. We implement all model internals ourselves as outlined below.

Core Architecture.

GPT-2 is a decoder-only transformer with $L = 12$ layers, hidden dimension $d = 768$, and $H = 12$ attention heads, initialized from OpenAI’s pretrained weights via HuggingFace. Each token is represented as the sum of a learned token embedding and a learned positional embedding. Each layer applies pre-norm residual blocks, where LayerNorm is applied before each sub-layer:

$$\mathbf{h} = \mathbf{x} + \text{Dropout}(W_o \cdot \text{sublayer}(\text{LayerNorm}(\mathbf{x})))$$

Causal Self-Attention.

Attention scores are scaled by head dimension $d_h = d/H$ and masked with a causal mask M_{causal} (to prevent attending to future tokens) and a padding mask M_{pad} . Then, outputs across all heads are concatenated and projected back to d :

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^\top}{\sqrt{d_h}} + M_{\text{causal}} + M_{\text{pad}}\right)V$$

Weight-Tied Output Projection.

Following GPT-2, the final hidden states are projected to vocabulary logits by reusing the word embedding matrix:

$$\text{logits} = \mathbf{h} \cdot E_{\text{word}}^\top$$

AdamW Optimizer.

We implement AdamW [4] with bias-corrected moment estimates and decoupled weight decay:

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

where $\hat{m}_t = m_t/(1 - \beta_1^t)$ and $\hat{v}_t = v_t/(1 - \beta_2^t)$ are the bias-corrected first and second moment estimates.

Task-Specific Heads.

All three tasks operate on the hidden state of the last non-padding token after the final transformer layer. We study three downstream tasks:

- **Sentiment Classification:** A dropout layer followed by a linear projection from d to the number of classes is applied to the last token’s hidden state. We compare two fine-tuning regimes: (1) *last-linear-layer*, where GPT-2 weights are frozen and only the classification head is trained, and (2) *full-model*, where all parameters are updated end-to-end.
- **Paraphrase Detection:** Each question pair is formatted as a cloze-style prompt: “*Question 1: ‘s1’, Question 2: ‘s2’, Are these questions asking the same thing?*” A 2-class linear head is applied to the last token’s hidden state and trained with cross-entropy loss.
- **Sonnet Generation:** The weight-tied projection is applied to all hidden states to produce per-token logits, enabling next-token prediction over the full sequence. Given token sequence $x_{1:T}$, the model is trained with cross-entropy to predict x_{t+1} from $x_{\leq t}$ at every position. At inference, we decode autoregressively using nucleus (top- p) sampling [5] with temperature scaling and optional top- k truncation.

LoRA: Low-Rank Adaptation.

Our main extension is LoRA [2], implemented directly in the project codebase without external PEFT libraries. Full fine-tuning updates all 125M GPT-2 parameters, which is memory-intensive and risks overwriting broadly useful pretrained representations. LoRA addresses this by keeping pretrained weights frozen and learning a low-rank update in parallel:

$$W = W_0 + \Delta W, \quad \Delta W = \frac{\alpha}{r} BA,$$

where r is the rank, $A \in \mathbb{R}^{r \times d}$ is initialized with Kaiming uniform, and $B \in \mathbb{R}^{d \times r}$ is initialized to zero so that $\Delta W = 0$ at the start of training. The scaling factor α/r controls the magnitude of the update relative to the rank.

In our implementation, LoRA modules wrap the self-attention query and value projection layers in every transformer block, while all other GPT-2 weights remain frozen as shown in Figure 1. This isolates adaptation to a small trainable subspace and makes the parameter budget explicit. For GPT-2 small with $L = 12$ layers and LoRA applied to query and value projections, the number of trainable parameters is $2 \times 2 \times d \times r \times L = 36,864 \times r$ (e.g., 147,456 at $r = 4$, versus 125,177,856 for full fine-tuning).

LoRA Integration Used in This Project (Sonnet + Paraphrase)

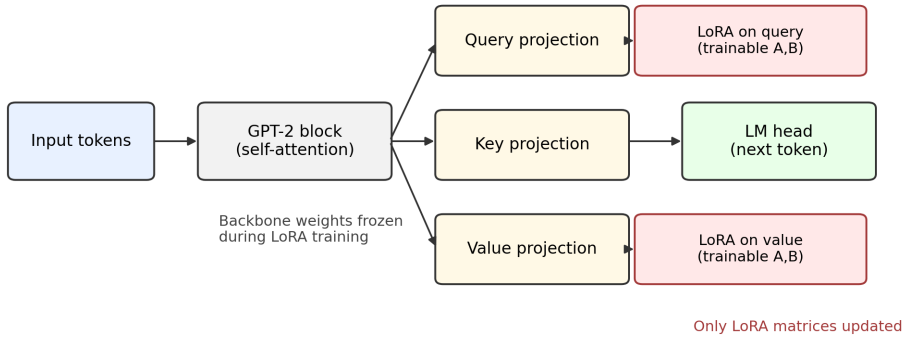


Figure 1: LoRA integration used in our project. We freeze GPT-2 backbone weights and train only low-rank adapters on the query and value projections of each self-attention block.

We apply LoRA to two tasks. For sonnet generation, we run a rank sweep over $r \in \{1, 2, 4, 8, 16\}$ to characterize how rank affects generation quality, comparing each against the full fine-tuning baseline using chrF. For paraphrase detection, we apply the same rank sweep over $r \in \{1, 2, 4, 8, 16\}$ to test whether the same integration strategy transfers to a classification setting and how rank affects accuracy. We keep the decoding settings fixed across all rank comparisons so that differences in results come from the training strategy rather than the sampling choices.

5 Experiments

5.1 Data

Sentiment classification. We use two datasets: SST [8], a 5-class movie review dataset (8,544/1,101 train/dev examples), and CFIMDB [9], a binary sentiment dataset (1,701/245 train/dev examples). **Paraphrase detection.** We use the Quora Question Pairs dataset [10] (141,506/20,215 train/dev pairs), where each example contains two questions and a binary label indicating whether they are paraphrases. **Sonnet generation.** We fine-tune on 143 Shakespearean sonnets. At evaluation, the model is given the first three lines of each of 12 held-out sonnets and must generate the continuation. We compute automatic quality against held-out dev references (TRUE_sonnets_held_out_dev.txt) and save generated outputs for qualitative inspection.

5.2 Evaluation Method

For sentiment classification and paraphrase detection, we report dev-set accuracy (fraction of correctly predicted labels). For sonnet generation, we report chrF [3], a character n -gram F-score that captures character-level style better than word-based metrics such as BLEU, and is more tolerant to stylistic variation in poetry generation. chrF computes character n -gram precision P and recall R and combines them as:

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

where P and R are averaged over character n -gram orders (we use $n = 1 \dots 6$, $\beta = 2$). Higher chrF indicates closer match to reference text.

5.3 Experimental Details

All models use our GPT-2 implementation (12 layers, hidden size 768, 12 heads, 125M parameters) initialized from OpenAI’s pretrained weights via HuggingFace [1]. All training is run on Modal-hosted NVIDIA T4.

Sentiment classification trains for 10 epochs with learning rate 1×10^{-3} (last-linear-layer) or 1×10^{-5} (full-model), batch size 8, and dropout 0.3. **Paraphrase detection** trains for 3 epochs with learning rate 1×10^{-5} and batch size 32. **Sonnet generation (full fine-tuning)** trains for 10 epochs with learning rate 1×10^{-5} , batch size 8, temperature 1.2, and nucleus sampling top- $p = 0.9$.

Sonnet generation (LoRA) uses the same task setup with learning rate 1×10^{-4} , 10 epochs, $\alpha = r$, and ranks $r \in \{1, 2, 4, 8, 16\}$ applied to query and value projections. Decoding settings are kept fixed across all ranks. **Paraphrase detection (LoRA)** follows the same cloze-style prompt setup and training configuration as the baseline (3 epochs, learning rate 1×10^{-5} , batch size 32), running a full rank sweep over $r \in \{1, 2, 4, 8, 16\}$ applied to query and value projections.

5.4 Results

Baseline results. Table 1 reports dev and test results across all tasks. For sentiment classification, our CFIMDB results meet or exceed the DFP baseline in both fine-tuning regimes: last-linear-layer achieves 0.878 (vs. 0.861) and full-model achieves 0.980 (vs. 0.976). SST results are more modest, with last-linear-layer slightly below baseline (0.453 vs. 0.462) and full-model matching it exactly (0.513). For paraphrase detection, our model achieves 0.885 dev accuracy under full fine-tuning, with a test leaderboard score of 0.830 (58th), which is consistent with generalization to unseen question

pairs. Sonnet generation is our strongest result: full fine-tuning achieves chrF = 32.5 on the dev set and chrF = 41.8 on the test leaderboard, placing us 17th overall. This suggests that our GPT-2 sonnet model generalizes beyond the held-out dev sonnets used during development.

Task	Mode	DFP Baseline	Dev	Test (Rank)
SST (5-class)	Last-linear-layer	0.462	0.453	—
SST (5-class)	Full-model	0.513	0.513	—
CFIMDB (binary)	Last-linear-layer	0.861	0.878	—
CFIMDB (binary)	Full-model	0.976	0.980	—
Quora (paraphrase)	Full fine-tune	—	0.885	0.830 (58th)
Sonnet generation	Full fine-tune	—	32.524	41.8 (17th)

Table 1: Dev and test results across all tasks. All classification tasks (SST, CFIMDB, paraphrase) report accuracy; sonnet generation reports chrF. Test leaderboard ranks are shown in parentheses.

LoRA results: sonnet generation. Table 2 shows the sonnet LoRA rank sweep. Quality generally improves with rank after $r = 2$, and $r = 16$ gives the best LoRA chrF of 28.892. Figure 2 visualizes the same trend and the quality–efficiency tradeoff. The key result is that LoRA reaches non-trivial generation quality while training fewer than 0.5% of parameters. Full fine-tuning still performs best in absolute chrF (32.524), but LoRA is competitive relative to its parameter budget.

Model	Trainable Params	Trainable %	Sonnet chrF
LoRA $r = 1$ (q,v)	36,864	0.029%	25.496
LoRA $r = 2$ (q,v)	73,728	0.059%	22.382
LoRA $r = 4$ (q,v)	147,456	0.118%	24.025
LoRA $r = 8$ (q,v)	294,912	0.236%	26.734
LoRA $r = 16$ (q,v)	589,824	0.471%	28.892
Full fine-tuning	125,177,856	100.000%	32.524

Table 2: LoRA rank sweep for sonnet generation (query/value adapters).

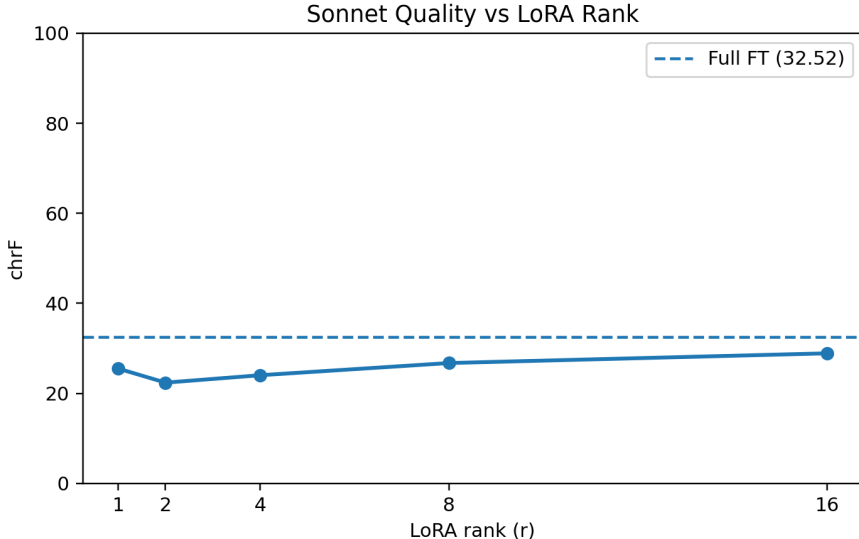


Figure 2: Sonnet dev chrF across LoRA ranks. Higher rank generally improves quality, with $r = 16$ best among LoRA settings.

LoRA results: paraphrase detection. Table 3 and Figure 3 show the full rank sweep for paraphrase detection. A rank–quality trend is present but weaker than in sonnet generation: accuracy improves

from 0.752 at $r = 1$ to 0.795 at $r = 16$, a gain of only 0.043 across the entire sweep. Full fine-tuning (0.885) remains substantially stronger than the best LoRA setting, with a gap of 0.090 even at $r = 16$.

Model	Trainable Params	Trainable %	Quora Dev Accuracy
LoRA $r = 1$ (q,v)	36,864	0.029%	0.752
LoRA $r = 2$ (q,v)	73,728	0.059%	0.762
LoRA $r = 4$ (q,v)	147,456	0.118%	0.779
LoRA $r = 8$ (q,v)	294,912	0.236%	0.782
LoRA $r = 16$ (q,v)	589,824	0.471%	0.795
Full fine-tuning	125,177,856	100.000%	0.885

Table 3: LoRA rank sweep for paraphrase detection (query/value adapters).

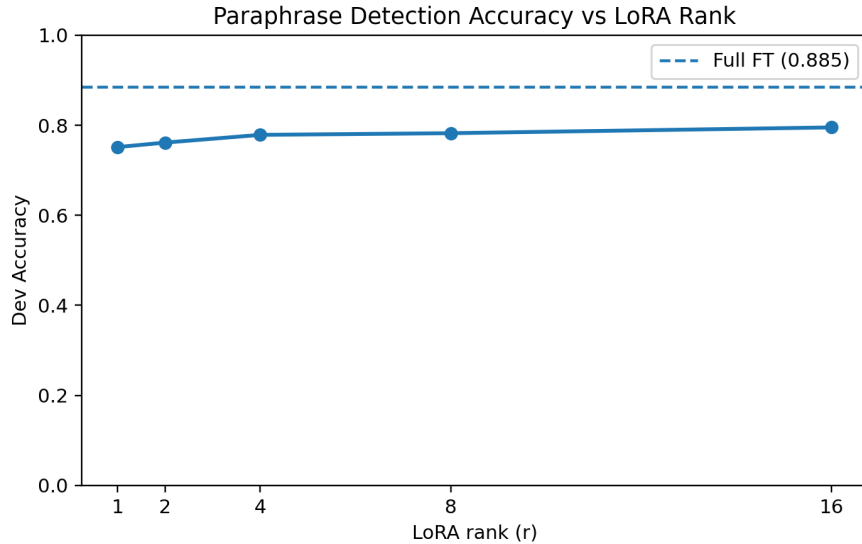


Figure 3: Paraphrase dev accuracy across LoRA ranks. Higher rank generally improves accuracy, with $r = 16$ best among LoRA settings.

6 Analysis

Rank sensitivity. The sonnet and paraphrase experiments both show that performance changes with rank, but the pattern differs across the two tasks. For sonnet generation, chrF drops to its lowest value at $r = 2$ (22.4) and then rises again through $r = 16$ (28.9), which suggests that higher ranks help the model better learn the style of Shakespearean verse. For paraphrase detection, the trend is smoother and more monotonic: accuracy improves steadily from 0.752 at $r = 1$ to 0.795 at $r = 16$, with diminishing gains at higher ranks. In both tasks, $r = 16$ achieves the best LoRA result, though neither closes the gap to full fine-tuning.

Cross-task behavior. The same LoRA integration transfers across tasks, but results differ substantially by task type. Even at $r = 16$, LoRA trains under 0.5% of parameters, yet the gap to full fine-tuning remains meaningful for both tasks: 3.6 chrF points for sonnet generation (11% relative degradation) and 0.090 accuracy points for paraphrase detection (10% relative degradation). Even though the relative gaps are similar, the loss matters more for classification. Paraphrase detection depends on making one correct binary decision, so it is less forgiving than generation where errors are spread across many tokens.

Qualitative patterns. To illustrate qualitative differences across ranks, we examine continuations of the same held-out sonnet prompt (“Poor soul, the center of my sinful earth / Pressed with these rebel powers that thee array / Why dost thou pine within and suffer dearth”):

LoRA $r = 1$: *“Popous mind thinks act the slow waste of beasts / In thy bountiful breast, as vale forty thou beset without gods eyes / As thou (even bee in the swimming pool), returning smell to the light...”*

LoRA $r = 16$: *“Effortless with hearses or small bombs thus sent / To conceal freedom? / ... Thou art pitiless, hopelessly consumed; / Nor is life hidden by fairest eyes...”*

Full fine-tuning: *“In witness of her mistress’s flowers, / I could do nothing’st in favor with thee but go home, / ... And, lo, such a darling sings jealously to me about thee...”*

At $r = 1$, the model produces text that is mostly incoherent and does not show clear meter or Shakespearean-style language. At $r = 16$, archaic phrasing (“Thou art”, “Nor is”) and some structural coherence emerge, though semantic consistency remains weak. Full fine-tuning yields more sustained poetic register and imagery. These examples suggest that rank controls how much the model can adapt. At very low rank, the model struggles to produce coherent or stylistically consistent verse. Higher ranks improve fluency, but full fine-tuning still does the best job matching the style of the training sonnets.

7 Conclusion

We implemented GPT-2 from scratch and fine-tuned it on three tasks, with baseline results meeting or exceeding DFP reference scores on CFIMDB. We then added LoRA and ran a rank sweep ($r \in \{1, 2, 4, 8, 16\}$) for both sonnet generation and paraphrase detection. In both tasks, $r = 16$ gives the best LoRA result while training under 0.5% of the model parameters. Overall, the results suggest that the quality–efficiency tradeoff of LoRA depends on the task. Both tasks incur roughly 10% relative degradation at $r = 16$, but the practical cost differs by task structure, with classification more sensitive to the low-rank constraint than generation.

This work has three main limitations. First, the sonnet training set is small (143 sonnets), which likely introduces variance in low-rank runs and makes it difficult to distinguish genuine rank effects from noise. Second, our LoRA implementation targets only the query and value projections in each attention block, leaving key projections and MLP layers unadapted; broader adapter coverage could close the remaining gap to full fine-tuning without substantially increasing the parameter budget. Third, all results are based on a single training seed, limiting the statistical robustness of our rank comparisons. Future work could address this by training across multiple seeds to measure variance, expanding LoRA to additional projection layers or feed-forward sublayers, and testing on larger GPT-2 variants to examine whether rank sensitivity scales with model capacity. On the evaluation side, chrF is an automatic proxy that may not fully capture poetic quality; complementing it with human judgments of fluency and Shakespearean style would provide a more complete picture of generation quality under different adaptation regimes.

Team Contributions

Both team members contributed equally and substantially to planning, experiments, and writing. Lucia led the milestone-stage baseline implementation, including the GPT-2 architecture, task-specific fine-tuning pipelines, and early evaluation setup. Nicole led the final-stage extension, implementing LoRA integration and running experiments across sonnet generation and paraphrase detection settings. Both authors collaborated on analysis, interpretation of results, and final report writing and editing.

References

- [1] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. *OpenAI Blog*, 1(8):9, 2019.
- [2] 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.
- [3] 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.

- [4] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *International Conference on Learning Representations*, 2019.
- [5] 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.
- [6] Angela Fan, Mike Lewis, and Yann Dauphin. Hierarchical neural story generation, 2018.
- [7] Dongkyu Lee, Gyeonghun Kim, Janghoon Han, Taesuk Hong, Yi-Reun Kim, Stanley Jungkyu Choi, and Nevin L. Zhang. Local temperature beam search: Avoid neural text DeGeneration via enhanced calibration. In *Findings of the Association for Computational Linguistics: ACL 2023*, pages 9903–9915, Toronto, Canada, 2023. Association for Computational Linguistics.
- [8] 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 EMNLP*, pages 1631–1642, 2013.
- [9] Andrew L Maas, Raymond E Daly, Peter T Pham, Dan Huang, Andrew Y Ng, and Christopher Potts. Learning word vectors for sentiment analysis. In *Proceedings of the 49th Annual Meeting of the ACL*, pages 142–150, 2011.
- [10] Shankar Iyer, Nikhil Dandekar, and Kornel Csernai. First Quora dataset release: Question pairs. <https://quoradata.quora.com/First-Quora-Dataset-Release-Question-Pairs>, 2017.