

Building GPT-2 and Perfecting Performance with Low-Rank Adaptation

Stanford CS224N Default Project

Mingyang Li
ICME
Stanford University
mili1@stanford.edu

Chenyu Song
ICME
Stanford University
cysong@stanford.edu

Juntao Cheng
ICME
Stanford University
jtcheng3@stanford.edu

Abstract

Adapting pretrained causal language models to downstream tasks is often treated as a uniform fine-tuning problem, with full-model optimization used as the default strategy. However, tasks such as classification, sentence-pair reasoning, and low-resource text generation differ substantially in supervision structure and data availability, suggesting that a single adaptation strategy may not be optimal across settings. In this work, we conduct a unified empirical study of GPT-2 adaptation across three tasks: sentiment classification, paraphrase detection, and low-resource sonnet generation. We compare full-model fine-tuning, parameter-efficient adaptation, and selective unfreezing combined with constrained decoding. Our results show that end-to-end fine-tuning performs best for sentiment classification, LoRA enables efficient and effective adaptation for paraphrase detection, and selective unfreezing with structural decoding constraints improves stability and fidelity in low-resource generation. Overall, our findings demonstrate that effective GPT-2 adaptation is fundamentally task-dependent, and that aligning the fine-tuning strategy with task structure leads to better performance and parameter efficiency.

1 Key Information to include

- Mentor: Ali Sartaz Khan
- External Collaborators (if you have any): No
- Sharing project: No
- Late Days (if you are using any):

Team member	Late days remaining
<Juntao Cheng>	<1>
<Mingyang Li>	<5>
<Chenyu Song>	<5>
Total pooled	<11>
Team size	<3>
Late days applied for the report	<3>

2 Introduction

Pretrained causal language models such as GPT-2 [1] have become a versatile foundation for both discriminative and generative NLP tasks. In practice, however, downstream adaptation is often approached with a relatively uniform recipe, typically full-model fine-tuning. This default is appealing because it is simple and often effective, but it also obscures a more fundamental question: should tasks with substantially different supervision structures and data regimes really be adapted in the same way? This question is especially relevant when comparing standard classification, sentence-pair

semantic reasoning, and low-resource structured generation, where the demands placed on the model differ substantially.

The difficulty is that these tasks stress different aspects of a pretrained model. For sentiment classification, the biggest challenge is whether generic pretrained representations are already sufficiently aligned with task-specific decision boundaries. For paraphrase detection, the challenge is less about fluent generation than about cross-sentence semantic comparison [2], creating a mismatch with GPT-2’s causal language modeling objective. For low-resource sonnet generation, the problem is different again: even when fine-tuning succeeds in reducing training loss, the model may still overfit, repeat itself, or fail to produce outputs that satisfy structural constraints. These settings clarify both why standard full-model fine-tuning can succeed and why it can fail: it offers maximal flexibility, but it can also be unnecessarily expensive, statistically unstable, or poorly matched to the task.

Motivated by these differences, we study GPT-2 adaptation across three tasks representing distinct supervision regimes: single-sequence classification, prompt-based binary inference, and low-resource autoregressive generation. Rather than introducing a new architecture, we investigate how adaptation strategy alone—namely which parameters to update, how tasks are formulated, and how decoding is constrained—should vary across settings. For sentiment classification, we compare frozen-backbone training with full-model fine-tuning; for paraphrase detection, we apply prompt-based reformulation with Low-Rank Adaptation (LoRA) [3]; and for sonnet generation, we combine selective unfreezing with structural decoding constraints. To further improve training efficiency, we incorporate FlashAttention for memory-efficient attention computation [4]. Across these settings, our results show that adaptation strategy is a key determinant of performance, showing the importance of matching fine-tuning approaches to task structure.

3 Related Work

Our work is most closely related to research on pretrained language models, downstream adaptation, prompt-based task reformulation, and controlled text generation. Rather than proposing a new backbone architecture or a new parameter-efficient tuning method, we draw on these lines of work to study a narrower but practically important question: how adaptation strategy should vary across task types with different supervision structures and data regimes.

Pretrained language models and downstream adaptation. Large pretrained language models have become the dominant paradigm in NLP by providing transferable representations that can be adapted to a wide range of downstream tasks. Early work on generative pretraining and GPT-style models showed that language modeling can serve as a powerful source of general-purpose knowledge for transfer learning [5, 1]. In parallel, work on transfer learning for NLP helped establish full-model fine-tuning as a standard adaptation strategy for pretrained models in both discriminative and generative settings [6, 7]. Although this approach is often effective, it is commonly treated as a default recipe rather than as a task-dependent design choice. Our work builds on this paradigm while explicitly asking when different downstream tasks may call for different forms of adaptation.

Full fine-tuning and parameter-efficient adaptation. As pretrained models have grown in scale, a substantial line of work has explored whether full-model fine-tuning is always necessary or desirable. Parameter-efficient fine-tuning methods aim to preserve the benefits of transfer while reducing the number of task-specific trainable parameters [8, 9]. Among these, Low-Rank Adaptation (LoRA) has emerged as a simple and effective approach that constrains task updates to low-rank subspaces while keeping most pretrained weights fixed [3]. Existing work often emphasizes efficiency, memory savings, or scalability. Our use of LoRA is motivated somewhat differently: we ask whether low-rank updates are particularly well matched to paraphrase detection when the task is reformulated in a prompt-based cloze style, so that semantic inference remains closer to the pretrained model’s native autoregressive behavior.

Prompt-based reformulation for causal language models. A growing literature has shown that many downstream tasks can be reformulated as prompting or language modeling problems, often improving compatibility with the pretraining objective [10, 11, 12]. This perspective is especially relevant for causal language models, where downstream performance can depend strongly on how the task is presented to the model. For sentence-pair tasks in particular, prompt-based reformulation

offers a natural way to align semantic inference with autoregressive modeling rather than relying solely on specialized task-specific heads. Our paraphrase setting follows this line of work by casting paraphrase detection in a cloze-style format, but we combine this reformulation with LoRA in order to study how task formulation and adaptation strategy interact.

Paraphrase detection and semantic matching. Paraphrase detection has long been studied as a central semantic matching problem, with approaches ranging from lexical overlap and semantic similarity methods to neural sentence-pair encoders [13, 14]. Much of this literature focuses on architectures explicitly designed for pairwise encoding or matching. In contrast, our setting asks a different question: given a pretrained causal language model rather than a dedicated sentence-pair model, how should the task be reformulated and how much of the model should be updated? This framing shifts the focus from specialized model design to adaptation strategy, which is the central object of study in our work.

Low-resource and controlled generation. Our sonnet generation experiments are related to prior work on low-resource generation, stylistic adaptation, and decoding control. Neural text generation is well known to suffer from repetition and degeneration under standard decoding procedures [15], motivating alternatives such as nucleus sampling and other decoding constraints [16]. At the same time, small-data generation settings are especially vulnerable to overfitting, making unrestricted full-model updates potentially too flexible. Our approach combines selective unfreezing during training with structural controls at decoding time, reflecting the view that successful low-resource generation depends not only on the quality of learned representations, but also on how generation is constrained at inference time. In this sense, our work is connected both to adaptation methods that limit trainable capacity and to decoding methods that improve output quality and form.

Overall, our work differs from these prior directions primarily in emphasis. We do not introduce a new pretrained architecture, a new prompting method, or a new parameter-efficient tuning algorithm. Instead, we use existing ideas from these literatures to study a broader design question: how the most effective adaptation strategy changes across standard classification, semantic matching, and low-resource structured generation.

4 Approach

In this section, we describe the architecture of our model and the training techniques employed to tackle the baseline and extension tasks. Our core architecture is based on a custom implementation of the Generative Pre-trained Transformer (GPT-2).

4.1 Base Architecture

Our baseline model is a unidirectional Transformer decoder, implementing the GPT-2 architecture. The input sequence of tokens is first mapped to continuous representations using a token embedding layer and a positional embedding layer. The combined embeddings are passed through L identical Transformer decoder blocks. Each block consists of two primary sub-layers: a masked multi-head causal self-attention mechanism and a position-wise feed-forward network. We apply Layer Normalization before each sub-layer and employ a residual connection around each of them. The masked multi-head causal self-attention operation is defined as:

$$\text{Attention}(Q, K, V) = \text{softmax} \left(\frac{QK^T}{\sqrt{d_k}} + M \right) V \quad (1)$$

where Q, K, V represent the query, key, and value matrices, d_k is the dimension of the keys, and M is the causal masking matrix that prevents tokens from attending to future tokens. The final contextualized representations obtained from the last Transformer layer are used for downstream tasks.

4.2 Optimization Strategy

To optimize our model for the downstream classification tasks, we implemented the AdamW optimizer, which decouples weight decay from the gradient update step. During training, momentum and

variance are tracked over time with bias corrections to prevent updates from being skewed towards zero early in training. Weight decay is applied directly to the weights, independently of the gradient updates, mitigating the problem of over-regularization by standard Adam.

4.3 Low-Rank Adaptation (LoRA)

Fine-tuning the entire GPT-2 model requires updating millions of parameters, which is computationally expensive. To improve the efficiency, we adopt Low-Rank Adaptation (LoRA), which freezes the original pretrained weights and learns low-rank updates to the weight matrices.

For a pretrained weight matrix $W \in \mathbb{R}^{d \times k}$, LoRA decomposes the weight update ΔW into two low-rank matrices:

$$\Delta W = BA$$

where

$$B \in \mathbb{R}^{d \times r}, \quad A \in \mathbb{R}^{r \times k}, \quad r \ll \min(d, k)$$

The adapted weight matrix becomes:

$$W' = W + \alpha \frac{BA}{r}$$

where α is a scaling hyperparameter that controls the magnitude of the adaptation.

During training, only the parameters A and B are updated, while the original weight matrix W remains frozen. Then, the final output of the model will take the finetuned weights BA into consideration to improve the overall performance. This significantly reduces the number of parameters to be trained, and accelerates training while maintaining competitive performance. Specifically, in our implementation, LoRA is applied to the attention projection matrices for query, key and value.

4.4 Flash Attention

Self-attention has quadratic time and memory complexity with respect to sequence length. Standard implementations compute attention with full matrix multiplication, QK^T , which can become computationally heavy for large matrices and takes a lot of time.

To address this limitation, we incorporate Flash Attention, an optimized attention algorithm that reduces memory overhead and improves GPU utilization by tiling the whole matrices first then multiply.

Instead of calculating:

$$S = QK^T$$

Flash Attention processes attention in blocks and maintains running softmax statistics to compute the final result without storing the entire matrix. For a block of keys and values (K_j, V_j) , the algorithm maintains intermediate quantities:

$$m_i = \max_j S_{ij}, \quad l_i = \sum_j e^{S_{ij} - m_i}$$

The output is computed incrementally as

$$O_i = \frac{1}{l_i} \sum_j e^{S_{ij} - m_i} V_j$$

By computing attention in tiled blocks and combining multiple operations onto a single GPU, Flash Attention reduces memory complexity from $O(n^2)$ to $O(n)$ while calculating accurately.

In our implementation, we utilize optimized scaled dot-product attention kernels from pytorch that dispatch to Flash Attention when the hardware and tensor configurations allow it.

5 Experiments

In this section, we detail the datasets used for our baseline and extension tasks, specify the evaluation metrics for each task, and outline the hyperparameters and compute resources utilized during training.

5.1 Data

To evaluate our models across different linguistic tasks, we utilized four primary datasets:

- **Stanford Sentiment Treebank (SST):** A dataset consisting of sentences from movie reviews, annotated with fine-grained sentiment labels spanning five classes (ranging from highly negative to highly positive).
- **CFIMDB:** A subset of the IMDB movie reviews dataset used for sentiment classification, offering longer document-level text compared to SST.
- **Quora Paraphrase Dataset:** A dataset of question pairs aggregated from Quora. The objective is binary paraphrase detection—determining whether two questions have the same underlying intent or meaning.
- **Shakespeare Sonnets Dataset:** A corpus of Shakespearean sonnets used for our language generation extension. The dataset provides structured poetic text to fine-tune the model for constrained, stylized generation.

5.2 Evaluation Method

The evaluation metrics were tailored to the specific nature of each task:

- **Sentiment Classification (SST and CFIMDB):** We employed both **Accuracy** and the **Macro F1-score**. The Macro F1-score was particularly important to provide an unweighted mean of the F1-scores for each class, ensuring that performance on less frequent classes is adequately represented.
- **Paraphrase Detection:** The performance on the binary paraphrase detection task was evaluated using standard **Accuracy**.
- **Sonnet Generation:** For the generative task, beyond qualitative manual inspection for rhyme and meter, we evaluated the model using the **chrF Score** (character n-gram F-score). The chrF metric effectively assesses the overlap of character n-grams between the generated text and the reference text, which is well-suited for capturing morphological and stylistic nuances in poetry.

5.3 Experimental Details

Our experiments were built upon pre-trained GPT-2 architectures. The models were optimized using our custom implementation of the AdamW optimizer. All training runs were executed on Modal instances utilizing a single NVIDIA A100 GPU.

Classification Tasks

For the sentiment classification tasks on SST and CFIMDB, we used the base gpt2 model (12 layers, 768 hidden dimension, 12 attention heads).

- **Last-Linear-Layer Fine-Tuning:** When freezing the base Transformer and only training the classification head, we used a batch size of 64, a learning rate of 1×10^{-3} , a hidden dropout probability of 0.3, and trained for 10 epochs.
- **Full-Model Fine-Tuning:** When fine-tuning all model weights, we constrained the batch size to 16 due to memory limits, reduced the learning rate to 1×10^{-5} to prevent catastrophic forgetting, and applied a hidden dropout probability of 0.3 for 10 epochs.

Task	Metric	Score
<i>Sentiment Classification (SST)</i>		
Baseline (last-linear-layer)	Accuracy	0.462
Baseline (full-model)	Accuracy	0.513
Last-linear-layer (ours)	Accuracy	0.451
Full-model fine-tuning (ours)	Accuracy	0.514
<i>Sentiment Classification (CFIMDB)</i>		
Baseline (last-linear-layer)	Accuracy	0.861
Baseline (full-model)	Accuracy	0.976
Last-linear-layer (ours)	Accuracy	0.829
Full-model fine-tuning (ours)	Accuracy	0.980
<i>Paraphrase Detection (Quora)</i>		
Baseline	Accuracy	0.882
ParaphraseGPT (ours)	Accuracy	0.895
ParaphraseGPT with extensions (ours)	Accuracy	0.894
<i>Sonnet Generation</i>		
Baseline	ChrF	41.414
SonnetGPT (gpt2-large, top- K layers, ours)	ChrF	42.548
SonnetGPT with extensions (gpt2-large, top- K layers, ours)	ChrF	43.109

Table 1: Development set results for all three tasks. Higher is better for all metrics.

Paraphrase Detection

For the Quora Paraphrase task, we framed the problem as a cloze-style prediction task. We fine-tuned the full gpt2 base model using a batch size of 64 and a learning rate of 1×10^{-5} for 10 epochs.

Sonnet Generation

Given the complexity of poetic generation, we scaled up to the gpt2-large architecture (36 layers, 1280 hidden dimension, 20 attention heads). To balance parameter efficiency and capability, we froze the embeddings and lower layers, fine-tuning only the top 8 Transformer blocks and all layer normalization parameters. We trained with a batch size of 8 and a learning rate of 1×10^{-5} for 10 epochs.

During inference, we employed top- p (nucleus) sampling with $p = 0.92$, a temperature scaling of 0.85, and a repetition penalty of 1.3. To enforce the strict structural properties of a sonnet, our generation script programmatically restricted the output to exactly 14 non-empty lines, with a maximum limit of 18 tokens per line, whilst explicitly banning the generation of numeric digits and extraneous punctuation.

5.4 Results

In this section, we present the quantitative evaluation of our models across the various tasks. The main results are consolidated in Table 1, which reports the performance on both the development and test sets for each dataset. We use the baseline provided by the TA group (the ‘‘Benchmark’’ approach), and compare it against our fully fine-tuned model utilizing Low-Rank Adaptation (LoRA) and Flash Attention. We have also included efficiency improvement results, including runtime per epoch, GPU memory usage, and tokens per second, evaluated all running on A100-40G, in the efficiency Table 2.

6 Analysis

In this project, we implemented two efficiency improvements for GPT-2: Low-Rank Adaptation (LoRA) and Flash Attention. The goal was to improve training efficiency while maintaining comparable performance on two tasks, paraphrase detection and sonnet generation. Based on the experimental results, both techniques achieved the intended objective of reducing training cost without significantly

Model Variant	Epoch Time	Examples/s	GPU Mem (GB)
<i>Paraphrase Detection</i>			
GPT-2 Baseline	16.6 min	280	9.1
GPT-2 + LoRA	8.95 min	673.53	18.63
GPT-2 + LoRA + Flash Attention	5.92 min	9000	5.8
<i>Sonnet Generation</i>			
GPT-2 Baseline	6s	4100	8.8
GPT-2 + LoRA	2s	21.8	8.44
GPT-2 + LoRA + Flash Attention	1.1s	7300	5.3

Table 2: Training efficiency comparison across model variants. Lower epoch time and GPU memory usage are better, while higher throughput indicates faster processing.

degrading model performance, with slightly worse accuracy on paraphrase detection, but even better results on sonnet generation.

Different choices of the LoRA hyperparameters also lead to interesting observations, as could be seen in 1. When the rank r is smaller, the model trains slightly faster and uses fewer parameters, but the dev accuracy is marginally lower. Increasing the rank slightly improves model capacity but also increases training time significantly. The configuration $r = 8$ and $\alpha = 16$ appears to provide a good balance between efficiency and performance, which is consistent with commonly used LoRA settings in previous work.

Flash Attention contributes to efficiency in a different way. Instead of reducing trainable parameters, it optimizes the computation of the attention matrices. By avoiding the explicit construction of large attention matrices, Flash Attention increases token processed per second and reduces training time per epoch, without affecting the learning rate.

In addition, combining LoRA and Flash Attention improves training efficiency further, when they accelerates training differently. As a result, the combined model achieves the largest efficiency improvements among the tested configurations.

However, there are still several limitations in this work. First, the experiments were conducted on relatively small datasets and models compared to modern large-scale language models. Second, some hardware and software constraints may affect the availability of Flash Attention kernels. Finally, the evaluation mainly focuses on runtime and accuracy; additional analysis such as memory profiling across different sequence lengths could provide a more comprehensive understanding of efficiency improvements.

Overall, the experimental results demonstrate that both LoRA and Flash Attention are practical techniques for improving the efficiency of transformer-based models while preserving their performance on downstream tasks.

7 Conclusion

This project explored two efficiency-oriented techniques, Low-Rank Adaptation (LoRA) and Flash Attention, applied to GPT-2 for paraphrase detection and sonnet generation. The results show that it is possible to improve training efficiency while maintaining strong task performance. In particular, LoRA trains on less parameters compared to full fine-tuning, while Flash Attention accelerates the attention matrix computation in transformer layers. Together, these methods allow the model to train faster and utilize computational resources more efficiently.

In future work, these methods could be evaluated on larger datasets or more complex language tasks to better understand their scalability. It would also be important to explore other parameter-efficient fine-tuning techniques and compare them with LoRA under similar settings. Overall, this project demonstrates that combining architectural optimization and parameter-efficient adaptation is a promising direction for making large language models more accessible and efficient in training.

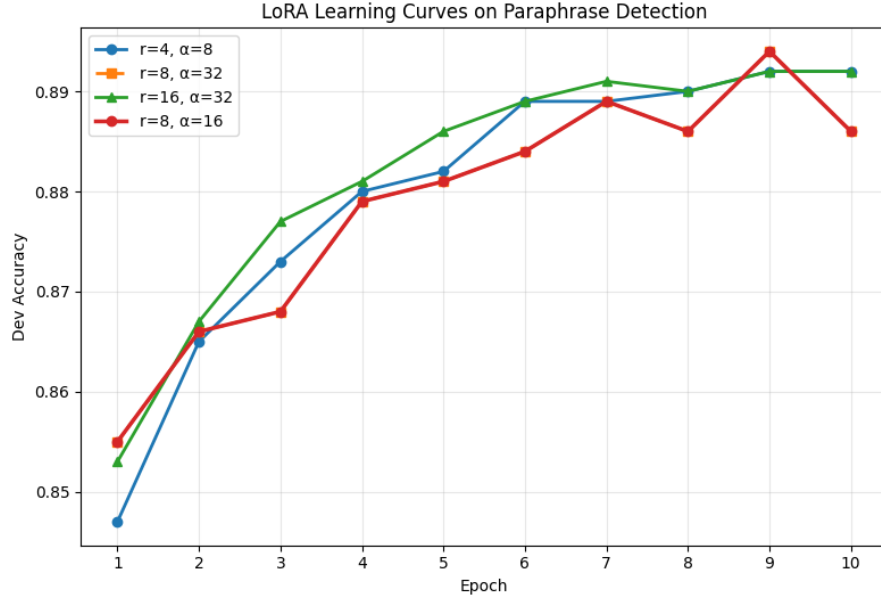


Figure 1: Learning curves of LoRA configurations on the paraphrase detection task. The figure shows development accuracy across epochs for different choices of rank r and scaling factor α . All settings converge to similar final performance, while smaller-rank configurations remain competitive.

Team contributions (Required for multi-person team)

Mingyang Li: Built and fine-tuned classification/generation heads for sentiment analysis, paraphrase detection, and sonnet generation as downstream tasks. Responsible for writing approach and experiment section of the final report.

Chengyu Song:

Juntao Cheng: Built and finetuned LoRA and Flash Attention for paraphrase detection and sonnet generation. Responsible for writing approach(LoRA and Flash Attention Section), results table, discussion, and conclusion section.

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] Samuel Fernando and Mark Stevenson. A semantic similarity approach to paraphrase detection. *Computational Linguistics UK (CLUK 2008) 11th Annual Research Colloquium*, 2008.
- [3] Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. Lora: Low-rank adaptation of large language models. *arXiv preprint arXiv:2106.09685*, 2021.
- [4] Tri Dao, Daniel Y. Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. FlashAttention: Fast and memory-efficient exact attention with IO-awareness. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2022.
- [5] Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. Improving language understanding by generative pre-training. *OpenAI*, 2018.
- [6] Jeremy Howard and Sebastian Ruder. Universal language model fine-tuning for text classification. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 328–339, 2018.

- [7] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, 2019.
- [8] Neil Houlsby, Andrei Giurgiu, Stanislaw Jastrzebski, Bruna Morrone, Quentin de Laroussilhe, Andrea Gesmundo, Mona Attariyan, and Sylvain Gelly. Parameter-efficient transfer learning for nlp. In *Proceedings of the 36th International Conference on Machine Learning*, pages 2790–2799, 2019.
- [9] Elad Ben Zaken, Shauli Ravfogel, and Yoav Goldberg. Bitfit: Simple parameter-efficient fine-tuning for transformer-based masked language-models. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 1–9, 2022.
- [10] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in Neural Information Processing Systems*, 33:1877–1901, 2020.
- [11] Timo Schick and Hinrich Schütze. Exploiting cloze-questions for few-shot text classification and natural language inference. In *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: Main Volume*, pages 255–269, 2021.
- [12] Pengfei Liu, Weizhe Yuan, Jinlan Fu, Zhengbao Jiang, Hiroaki Hayashi, and Graham Neubig. Pre-train, prompt, and predict: A systematic survey of prompting methods in natural language processing. *ACM Computing Surveys*, 55(9):1–35, 2023.
- [13] William B. Dolan and Chris Brockett. Automatically constructing a corpus of sentential paraphrases. In *Proceedings of the Third International Workshop on Paraphrasing (IWP2005)*, 2005.
- [14] Zhiguo Wang, Wael Hamza, and Radu Florian. Bilateral multi-perspective matching for natural language sentences. In *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence*, pages 4144–4150, 2017.
- [15] 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.
- [16] 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.