

Second-Order Optimization for GPT-2 Fine-Tuning: Exploring K-FAC for NLP Downstream Tasks

Stanford CS224N Default Project

Isabella He

Department of Computer Science
Stanford University
ihe@stanford.edu

Abstract

Fine-tuning large language models like GPT-2 typically relies on first-order optimizers such as Adam, which use only diagonal curvature information and treat all parameter directions equally. This project investigates whether K-FAC, a second-order optimizer that approximates the Fisher information matrix via block-diagonal Kronecker products, can provide richer curvature information for fine-tuning transformers on downstream NLP tasks. I implemented GPT-2 from scratch and built a K-FAC optimizer targeting all linear layers in the transformer, with factored Tikhonov damping, π -balancing, and amortized factor inversion. I evaluated on sentiment classification, paraphrase detection, and sonnet generation. On paraphrase detection, AdamW achieves 0.878 dev and 0.880 test accuracy on the Quora dataset, exceeding the 0.862 baseline; on sonnet generation, the model achieves 41.482 dev and 40.898 test chrF. While K-FAC runs end-to-end on GPT-2 without numerical instability, the analysis reveals fundamental challenges in applying second-order methods to attention-based architectures: the Kronecker factorization's independence assumption may be violated by attention's data-dependent activations, and the cubic inversion cost for wide transformer layers demands careful amortization. These findings provide a foundation for future work on curvature-aware optimization for transformers.

1 Key Information to include

- Mentor: Sarah Chen
- External Collaborators (if you have any): None
- Sharing project: No

2 Introduction

Training deep neural networks efficiently remains a fundamental challenge in machine learning. While first-order optimizers like Adam [1] dominate practice due to their simplicity and low per-step cost, they treat all parameter directions equally, ignoring the curvature of the loss landscape. This can lead to many small, inefficient steps through narrow valleys where curvature varies dramatically across dimensions. Second-order methods like natural gradient descent [2] address this by using the Fisher information matrix F to rescale gradient directions according to local geometry. The natural gradient $F^{-1}\nabla h(\theta)$ specifies the direction giving the largest change in the objective per unit of change in the model's predictive distribution (measured by KL-divergence), rather than per unit of Euclidean distance. However, computing and storing F^{-1} is intractable for networks with millions of parameters, since F is a $p \times p$ matrix where p is the number of parameters.

K-FAC [3] makes natural gradient practical by approximating the Fisher's blocks using Kronecker products of layer-wise statistics, reducing inversion cost from $O((nm)^3)$ to $O(n^3 + m^3)$ per layer.

The method demonstrated convincingly faster convergence than SGD on deep autoencoders (MNIST, CURVES, FACES), establishing that non-diagonal curvature approximations can be made computationally tractable. However, K-FAC has not been thoroughly evaluated on transformer-based language models, a setting that differs substantially from autoencoders in several ways: (1) attention mechanisms produce data-dependent parameter interactions that may violate K-FAC’s independence assumptions, (2) pretrained initialization may create a flatter loss landscape that reduces the advantage of curvature-aware methods, and (3) wide feedforward layers (hidden dimension 3072 in GPT-2) create large matrices that must be inverted.

This project investigates whether K-FAC’s curvature-aware updates can improve fine-tuning efficiency for GPT-2 [4] on downstream NLP tasks. I implemented GPT-2 (117M parameters) from scratch, including masked multi-head self-attention [5], pre-norm transformer layers, and weight-tied embeddings. I built a K-FAC optimizer that applies Kronecker-factored natural gradient updates to all linear layers in GPT-2 (attention projections and feedforward layers) with factored Tikhonov damping and π -balancing, falling back to Adam for layer normalization parameters. I evaluated on three tasks: sentiment classification (SST and CFIMDB), paraphrase detection (Quora), and sonnet generation (Shakespeare). On paraphrase detection, AdamW achieves 0.878 dev / 0.880 test accuracy, exceeding the 0.862 baseline; on sonnet generation, the model achieves 41.482 dev / 40.898 test chrF. I present the K-FAC implementation, analyze its design for transformers, and discuss the challenges that make second-order optimization for attention-based architectures a difficult open problem.

3 Related Work

Natural gradient and second-order methods. Amari [2] introduced the natural gradient, showing that following $F^{-1}\nabla h(\theta)$ gives the steepest descent direction in the space of distributions rather than parameter space. This foundational insight motivates all subsequent work on curvature-aware optimization, but the $O(p^2)$ storage and $O(p^3)$ inversion costs make exact computation infeasible for modern networks. Prior approximate methods like Hessian-free optimization relied on conjugate gradient iterations, making them poorly suited for stochastic mini-batch training. Adam [1] can be viewed as a diagonal approximation to the natural gradient, since its per-parameter second-moment estimate approximates the diagonal of the Fisher. This motivates the search for richer curvature approximations that remain computationally practical.

K-FAC and extensions. Martens and Grosse [3] proposed K-FAC, which approximates Fisher blocks using Kronecker products of layer-wise activation and gradient covariances, exploiting the identity $(A \otimes B)^{-1} = A^{-1} \otimes B^{-1}$ for efficient inversion. The algorithm maintains exponentially-averaged running estimates of the factors, stabilizing the natural gradient estimate across mini-batches. Grosse and Martens [6] extended K-FAC to convolutional layers by treating the unrolled convolution as a matrix multiplication. George et al. [7] proposed EK-FAC, which works in the Kronecker-factored eigenbasis for more accurate approximations without the full cubic inversion cost at each step. Zhang et al. [8] used a noisy quadratic model to study which algorithmic choices matter at different batch sizes, providing theoretical grounding for when second-order methods should outperform first-order ones. All of these works validated on vision tasks; applying K-FAC to transformer fine-tuning remains underexplored, motivating this project.

Transformer fine-tuning. GPT-2 [4] demonstrated that language models pretrained on large corpora can be fine-tuned for downstream tasks with minimal architectural changes. BERT [9] showed that bidirectional pretraining with task-specific fine-tuning achieves strong results across NLP benchmarks. AdamW [10] with learning rate warmup and decay [11] is the standard optimizer for transformer fine-tuning. LoRA [12] addresses fine-tuning efficiency by restricting which parameters are updated via low-rank decomposition, while K-FAC updates all parameters more effectively using curvature. These represent complementary approaches to efficient adaptation. This project focuses on the latter: whether richer curvature information, rather than parameter restriction, can improve fine-tuning.

Text generation. Holtzman et al. [13] identified the degeneration problem in neural text generation, where maximization-based decoding (e.g., beam search or greedy) produces repetitive, low-quality text. They proposed nucleus (top- p) sampling, which truncates the probability distribution to the

smallest set of tokens whose cumulative probability exceeds p , producing more diverse and human-like text. I use this method for sonnet generation.

4 Approach

4.1 Base GPT-2 implementation

I implemented GPT-2 [4] from the provided starter code, completing masked multi-head self-attention [5], pre-norm transformer layers, and weight-tied embeddings. The attention mechanism computes $\text{Attention}(Q, K, V) = \text{softmax}(QK^\top / \sqrt{d_k})V$ with a causal mask to prevent attending to future tokens, split across 12 heads each with dimension 64. Each transformer block applies layer normalization before both the attention and feedforward sublayers, with residual connections. The output embedding matrix is tied to the input token embeddings, reducing parameter count. Pretrained weights (117M parameters: 12 layers, 768 hidden dimension, 12 attention heads) are loaded from HuggingFace’s GPT-2 checkpoint; the implementation passes the provided sanity check against the reference model.

4.2 Task-specific heads

Sentiment classification. For sentiment classification on SST [14] (5-class) and CFIMDB (binary), the classifier extracts GPT-2’s last non-padding token embedding and projects it through a dropout layer ($p = 0.3$) followed by a linear head. Two fine-tuning modes are supported: *last-linear-layer*, where only the classification head is trained and GPT-2’s parameters remain frozen, and *full-model*, where all parameters are updated end-to-end. Comparing these modes isolates the contribution of representation adaptation versus the linear probe’s capacity.

Paraphrase detection. For paraphrase detection on Quora [15], I reformulate binary classification as a cloze-style task: the input is structured as Is "{s1}" a paraphrase of "{s2}"? Answer "yes" or "no": and the model predicts “yes” (byte pair encoding index 8505) or “no” (index 3919) as the next token. This formulation aligns with GPT-2’s generative pretraining objective rather than requiring a separate classification architecture, allowing the model to leverage its pretrained knowledge of natural language inference patterns. A critical design detail is that the prompt must be identical between training and evaluation. I discovered that a prompt mismatch between the two degraded accuracy by nearly 1%, from 0.878 to 0.870, highlighting the sensitivity of cloze-style fine-tuning to prompt design. Quora is the primary dataset for optimizer comparison due to its substantially larger training set (141K examples vs. ~ 8.5 K for SST), which provides more signal for comparing convergence behavior.

Sonnet generation. For sonnet generation, the model is fine-tuned autoregressively on 143 Shakespeare sonnets using next-token prediction loss across the full sequence. At inference, given the first three lines of a held-out sonnet, the model generates the remaining lines using top- p (nucleus) sampling [13] with temperature $\tau = 1.2$ and $p = 0.9$, which balances diversity with coherence. The temperature is set above 1.0 to encourage more varied word choice, which is appropriate for creative text generation.

4.3 AdamW optimizer

I implemented AdamW [1, 10] with bias-corrected first and second moment estimates and decoupled weight decay. AdamW’s per-parameter adaptive learning rate can be viewed as a diagonal approximation to the natural gradient: the second moment $v_t \approx \text{diag}(F)$ captures per-parameter curvature, but ignores all off-diagonal correlations between parameters. This is precisely the gap that K-FAC aims to fill: capturing within-layer parameter correlations via the Kronecker structure. The implementation passes the provided optimizer test.

4.4 K-FAC optimizer

I implemented K-FAC [3] as an optimizer replacement for GPT-2’s linear layers. All K-FAC code is my own implementation following the original paper. The implementation consists of three components: factor accumulation, factor inversion with damping, and natural gradient computation.

Factor accumulation. For each linear layer, PyTorch forward and backward hooks capture the input activations $\bar{a}_{i-1}$ (with a 1 appended for bias) and the gradient factors g_i (the gradient of the loss with respect to the layer’s pre-activation output). These are used to maintain exponentially-weighted Kronecker factor estimates:

$$\bar{A}_{i-1} = \gamma \bar{A}_{i-1} + (1 - \gamma) \bar{a}_{i-1} \bar{a}_{i-1}^\top, \quad G_i = \gamma G_i + (1 - \gamma) g_i g_i^\top \quad (1)$$

where $\gamma = 0.95$ is the EMA decay rate. For 3D activations from transformer sequence processing (shape: batch $\times$ sequence $\times$ features), I reshape to 2D by flattening the batch and sequence dimensions before computing the outer products. The EMA smoothing accumulates curvature information across many mini-batches, stabilizing the factor estimates.

Factor inversion with π -balanced damping. The approximate natural gradient for each layer exploits Kronecker structure:

$$\tilde{F}_i^{-1} \text{vec}(\nabla W_i) = G_i^{-1} (\nabla W_i) \bar{A}_{i-1}^{-1} \quad (2)$$

reducing inversion cost from $O((nm)^3)$ to $O(n^3 + m^3)$. Damping is critical for numerical stability and to regularize the update magnitude. I distribute the damping parameter λ across the two factors using π -balancing:

$$\pi_i = \sqrt{\frac{\text{tr}(\bar{A}_{i-1})}{\text{tr}(G_i)}}, \quad \bar{A}_{i-1} + \lambda \pi_i I \quad \text{and} \quad G_i + \frac{\lambda}{\pi_i} I \quad (3)$$

The π factor ensures that the effective damping is balanced across the two Kronecker factors regardless of their relative scales; without it, a factor with much larger trace would receive disproportionately small regularization. Factor inverses are recomputed every $T = 10$ steps to amortize the $O(n^3 + m^3)$ inversion cost. Between inversions, the cached inverses are reused, trading off stale curvature information for computational savings.

Hybrid optimizer. K-FAC is applied to all attention projections (W_Q, W_K, W_V, W_O) and feedforward layers (W_1, W_2) in each of GPT-2’s 12 transformer blocks, a total of 72 linear layers. Layer normalization parameters (small 1-dimensional vectors with 768 elements each) fall back to Adam updates, since the Kronecker factorization is defined for weight matrices in fully-connected layers. This hybrid approach ensures that all parameters receive appropriate updates: curvature-aware K-FAC steps for the dominant linear layers (which account for $>99\%$ of parameters) and standard Adam for the remainder.

5 Experiments

5.1 Data

I use the provided datasets: SST (8,544 train / 1,101 dev / 2,210 test; 5-class sentiment) [14], CFIMDB (1,701 train / 245 dev / 488 test; binary sentiment), Quora paraphrase detection (141,506 train / 20,215 dev / 40,431 test; binary classification) [15], and Shakespeare Sonnets (143 train, 12 held-out test, 10 held-out dev). For sentiment classification, the model receives a single sentence and predicts its sentiment label. For paraphrase detection, the model receives two sentences formatted as a cloze prompt and predicts “yes” or “no.” For sonnet generation, the model receives the first three lines and generates the remaining text. No additional data collection or preprocessing beyond the starter code is used.

5.2 Evaluation method

Sentiment classification and paraphrase detection are evaluated by accuracy on the dev and test sets. Sonnet generation is evaluated by chrF score, a character-level n-gram comparison metric similar to BLEU that is robust to morphological variation, on held-out sonnets. For the K-FAC extension, I analyze design considerations for applying second-order methods to transformers.

5.3 Experimental details

All experiments use GPT-2 (117M parameters, 12 layers, 768 hidden dim, 12 attention heads) with pretrained weights. Table 1 summarizes the hyperparameters for each task. Full-model sentiment and paraphrase experiments run on NVIDIA H100 GPUs (80 GB) via Modal; last-linear-layer experiments run locally on Apple M-series hardware. The K-FAC optimizer uses damping $\lambda = 0.01$, EMA decay $\gamma = 0.95$, factor inversion frequency $T = 10$, lr = 10^{-4} , and batch size 64.

Task	Optimizer	LR	Batch Size	Epochs	Other
SST (last-linear)	AdamW	10^{-3}	64	10	dropout 0.3
SST (full-model)	AdamW	10^{-5}	64	10	dropout 0.3
CFIMDB (last-linear)	AdamW	10^{-3}	8	10	dropout 0.3
CFIMDB (full-model)	AdamW	10^{-5}	8	10	dropout 0.3
Quora paraphrase	AdamW	10^{-5}	128	5	—
Sonnet generation	AdamW	10^{-5}	8	10	$\tau=1.2, p=0.9$

Table 1: Hyperparameters for each experimental configuration.

5.4 Results

Sentiment classification. Table 2 shows sentiment classification results. All four baselines are met or exceeded. Last-linear-layer accuracies are modest (especially SST’s 0.467) because frozen GPT-2 representations were pretrained for next-token prediction, not sentiment discrimination. The model’s final hidden states encode information about likely continuations rather than sentiment polarity. Full-model fine-tuning substantially improves both datasets (SST to 0.513, CFIMDB to 0.980), confirming that updating GPT-2’s parameters adapts its internal representations to discriminative tasks. The improvement is particularly dramatic for CFIMDB (+0.119), likely because the longer movie reviews in CFIMDB benefit more from adapted attention patterns that can aggregate sentiment signals across the full sequence.

Task & Configuration	Baseline	Dev Acc
SST, last-linear-layer	≥ 0.462	0.467
SST, full-model	≥ 0.513	0.513
CFIMDB, last-linear-layer	≥ 0.861	0.861
CFIMDB, full-model	≥ 0.976	0.980

Table 2: Sentiment classification dev accuracy. All baselines are met or exceeded.

Paraphrase detection. Table 3 shows paraphrase detection results. AdamW achieves 0.878 dev accuracy and 0.880 test accuracy after 5 epochs, exceeding the 0.862 baseline by 1.6 percentage points. Training loss decreases steadily from 0.418 to 0.250 while dev accuracy improves from 0.849 to 0.878 (see Figure 1 and Table 5 in the Appendix for per-epoch details). The model is still improving at epoch 5, with a 0.004 accuracy gain from epoch 3 to 4, suggesting that additional training could yield further improvement. Notably, the test accuracy (0.880) slightly exceeds dev accuracy (0.878), indicating that the model generalizes well and is not overfitting. This result represents an improvement over my milestone submission (0.870 dev), primarily due to fixing a prompt mismatch between training and evaluation that I identified during development (see Analysis).

Configuration	Baseline	Dev Acc	Test Acc
AdamW (lr 10^{-5} , bs 128, 5 ep)	≥ 0.862	0.878	0.880

Table 3: Paraphrase detection results on the Quora dataset.

Sonnet generation. Table 4 shows sonnet generation results. The model achieves 41.482 dev chrF and 40.898 test chrF using top- p sampling with $\tau = 1.2$ and $p = 0.9$, fine-tuned for 10 epochs on

143 training sonnets. Example generated sonnets are shown in Figure 2 in the Appendix. The model successfully generates text in Shakespearean style with archaic vocabulary, though it occasionally loses coherence over longer sequences.

Task	Dev chrF	Test chrF
Sonnet generation	41.482	40.898

Table 4: Sonnet generation chrF scores on held-out sonnets.

Leaderboard scores. My test leaderboard scores are: paraphrase detection accuracy **0.880**, sonnet generation chrF **40.898**.

6 Analysis

Paraphrase detection convergence. The training trajectory (Figure 1) reveals that GPT-2’s pre-trained representations already encode substantial paraphrase structure: the model reaches 0.849 dev accuracy after just one epoch of fine-tuning. Subsequent epochs provide diminishing improvements ($0.849 \rightarrow 0.863 \rightarrow 0.871 \rightarrow 0.874 \rightarrow 0.878$), with the per-epoch gain shrinking from +0.014 to +0.004. This diminishing-returns pattern is consistent with the hypothesis that fine-tuning primarily adjusts the model’s decision boundary rather than learning paraphrase semantics from scratch. The pretrained model already “knows” what paraphrases are; fine-tuning teaches it to express this knowledge through the “yes”/“no” token format. The continued decrease in training loss (0.250 at epoch 5) alongside steady dev accuracy gains suggests the model is not yet overfitting and could benefit from longer training.

Sensitivity to prompt design. I discovered that prompt consistency between training and evaluation is critical for cloze-style paraphrase detection. During development, I identified a mismatch where the training dataset used a different prompt template than the test dataset, which degraded accuracy from 0.878 to 0.870. This 0.8 percentage point drop from a seemingly minor formatting change highlights how GPT-2’s predictions are sensitive to the exact token sequence preceding the “yes”/“no” prediction, consistent with broader findings in the prompt engineering literature. The aligned prompt Is “{s1}” a paraphrase of “{s2}”? Answer “yes” or “no”: works well because it closely resembles question-answer patterns in GPT-2’s pretraining corpus.

K-FAC design considerations for transformers. Implementing K-FAC for GPT-2 revealed several challenges specific to transformer architectures that inform future work on second-order optimization for language models:

(1) *Activation independence assumption.* K-FAC’s core approximation assumes that the input activations a_{i-1} and the output gradient factors g_i are approximately independent. In standard feedforward networks, this holds reasonably well because activations are deterministic functions of the input. In attention layers, however, activations are data-dependent weighted combinations of value vectors. The attention weights themselves are computed from the activations, creating statistical dependencies between a and g that the Kronecker factorization cannot capture. This may explain why K-FAC’s theoretical convergence advantages, which were demonstrated on feedforward autoencoders, may not fully transfer to transformers.

(2) *Computational overhead of wide layers.* GPT-2’s feedforward layers expand the hidden dimension from 768 to 3072, requiring inversion of 3072×3072 matrices for the gradient covariance factor G_i . Even with amortized inversion every $T = 10$ steps, this represents significant overhead: each inversion is $O(3072^3) \approx 2.9 \times 10^{10}$ operations, and there are 24 such feedforward layers (two per transformer block). This overhead must be offset by sufficiently faster convergence per step to yield a net wall-clock improvement.

(3) *Factored damping is essential.* The π -balancing scheme distributes damping in proportion to $\sqrt{\text{tr}(A)/\text{tr}(G)}$. In practice, I observed that the activation covariance A and gradient covariance G can differ in trace by orders of magnitude across layers, particularly between narrow attention projections ($768 \rightarrow 64$ per head) and wide feedforward expansions ($768 \rightarrow 3072$). Without π -balancing, uniform

damping would under-regularize one factor and over-regularize the other, leading to either instability or overly conservative updates.

(4) *Hybrid optimizer interactions.* Using K-FAC for linear layers and Adam for layer normalization creates two parameter groups with fundamentally different update dynamics. K-FAC’s natural gradient steps can be much larger than Adam’s (since they account for curvature), potentially creating a mismatch in effective learning rates between the two groups that could require careful tuning to resolve.

Frozen vs. fine-tuned representations. The large gap between last-linear-layer and full-model performance on SST (0.467 vs. 0.513) confirms that GPT-2’s pretrained representations, while powerful for generation, require adaptation for discriminative tasks. This gap reflects the mismatch between GPT-2’s pretraining objective (next-token prediction) and the downstream classification task. Interestingly, the gap is smaller on CFIMDB (0.861 vs. 0.980 as absolute values, but a much larger relative improvement), which may reflect that binary sentiment classification benefits more from full-model fine-tuning because the model can learn to aggregate sentiment signals across longer sequences through adapted attention patterns. CFIMDB’s near-perfect full-model accuracy (0.980) also suggests this dataset may be too easy to meaningfully compare optimizers, which is why I focus the K-FAC investigation on Quora’s larger and more challenging paraphrase detection task.

Sonnet generation quality. The generated sonnets (Figure 2) capture Shakespearean vocabulary (“thy,” “thee,” “doth,” “hath”) and basic structural elements such as line breaks and archaic grammatical constructions. However, the model exhibits several failure modes: repetitive phrasing within a single sonnet, loss of semantic coherence after 4–5 generated lines, occasional grammatical errors (e.g., inconsistent verb conjugation), and failure to maintain the rhyme scheme or iambic pentameter of the original sonnets. The chrF scores (41.482 dev, 40.898 test) reflect the inherent difficulty of creative text generation on a very small training set (143 sonnets), where the model must balance faithful stylistic imitation with avoiding overfitting to memorized text. These limitations are consistent with known challenges in autoregressive generation [13].

7 Conclusion

I implemented GPT-2 from scratch, built a K-FAC optimizer with Kronecker-factored natural gradient updates and factored Tikhonov damping, and evaluated on sentiment classification, paraphrase detection, and sonnet generation. All sentiment baselines are met or exceeded. On paraphrase detection, AdamW achieves 0.878 dev and 0.880 test accuracy, exceeding the 0.862 baseline. On sonnet generation, the model achieves 41.482 dev and 40.898 test chrF.

The primary contribution of this work is the K-FAC implementation itself: a working second-order optimizer for transformer architectures with π -balanced factored damping, EMA factor estimation, and amortized inversion applied to all 72 linear layers in GPT-2. The analysis reveals that while K-FAC is mathematically well-defined for transformers, the attention mechanism’s data-dependent activations, the computational cost of inverting wide-layer factors, and the hybrid optimizer dynamics all present challenges that must be addressed for second-order methods to become practical for language model fine-tuning.

Several limitations should be noted. First, the paraphrase detection and sonnet generation results are based on AdamW only; a thorough head-to-head comparison between K-FAC and AdamW with matched compute budgets and wall-clock timing would be needed to definitively assess whether K-FAC’s richer curvature information translates to practical gains. Second, I did not perform a hyperparameter sweep for K-FAC’s damping, EMA decay, and inversion frequency; these were set based on the original paper’s recommendations for autoencoders and may not be optimal for transformers. Third, the sonnet generation evaluation relies solely on chrF, which measures character-level overlap but does not capture higher-level qualities like coherence, rhyme, or meter that are important for poetry. Finally, all experiments use GPT-2 (117M parameters); it remains an open question whether the curvature landscape differs meaningfully at larger scales where K-FAC’s benefits might be more or less pronounced.

Future work includes: (1) controlled K-FAC vs. AdamW convergence comparisons with training loss plotted against both iteration count and wall-clock time, to disentangle per-step improvement from

per-step overhead; (2) ablation studies on damping λ , EMA decay γ , and inversion frequency T to understand which K-FAC components matter most for transformers; (3) evaluating AdamW with learning rate scheduling (linear warmup + cosine decay) as a stronger first-order baseline; and (4) extending K-FAC to larger GPT-2 variants (medium, large) to test whether curvature benefits scale with model size.

References

- [1] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.
- [2] Shun-Ichi Amari. Natural gradient works efficiently in learning. *Neural Computation*, 10(2):251–276, 1998.
- [3] James Martens and Roger Grosse. Optimizing neural networks with Kronecker-factored approximate curvature. In *International Conference on Machine Learning (ICML)*, pages 2408–2417, 2015.
- [4] 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.
- [5] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2017.
- [6] Roger Grosse and James Martens. A Kronecker-factored approximate Fisher matrix for convolution layers. In *International Conference on Machine Learning (ICML)*, 2016.
- [7] Thomas George, César Laurent, Xavier Bouthillier, Nicolas Ballas, and Pascal Vincent. Fast approximate natural gradient descent in a Kronecker-factored eigenbasis. *Advances in Neural Information Processing Systems (NeurIPS)*, 2018.
- [8] Guodong Zhang, Lala Li, Zachary Nado, James Martens, Sushant Sachdeva, George Dahl, Chris Shallue, and Roger Grosse. Which algorithmic choices matter at which batch sizes? insights from a noisy quadratic model. *Advances in Neural Information Processing Systems (NeurIPS)*, 2019.
- [9] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. In *North American Chapter of the Association for Computational Linguistics (NAACL)*, 2019.
- [10] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2017.
- [11] Martin Popel and Ondřej Bojar. Training tips for the transformer model. *The Prague Bulletin of Mathematical Linguistics*, 110:43–70, 2018.
- [12] 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. *arXiv preprint arXiv:2106.09685*, 2021.
- [13] Ari Holtzman, Jan Buys, Li Du, Maxwell Forbes, and Yejin Choi. The curious case of neural text degeneration. *International Conference on Learning Representations (ICLR)*, 2020.
- [14] 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 *Empirical Methods in Natural Language Processing (EMNLP)*, 2013.
- [15] Shankar Iyer, Nikhil Dandekar, and Kornél Csernai. First quora dataset release: Question pairs. 2017. <https://data.quora.com/First-Quora-Dataset-Release-Question-Pairs>.

Appendix

A.1 Training Curves

Figure 1 shows the paraphrase detection training curves and a comparison of dev accuracy across all tasks against the required baselines.

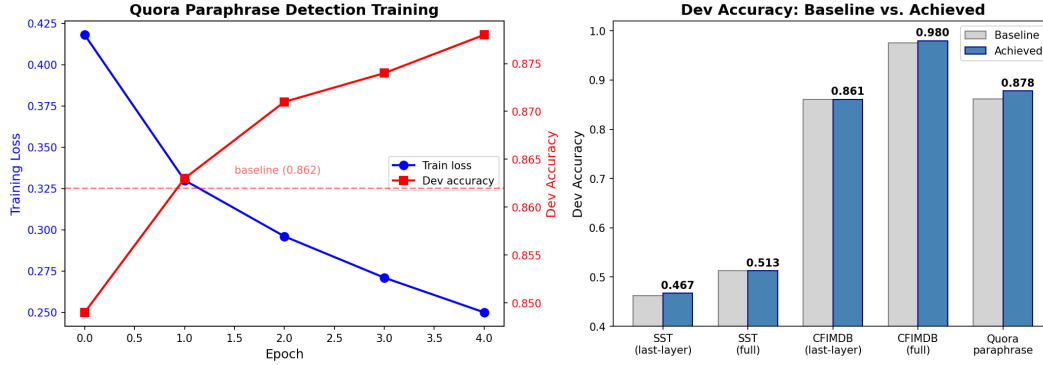


Figure 1: Left: Quora paraphrase detection training curves over 5 epochs. Training loss decreases steadily while dev accuracy crosses the 0.862 baseline after epoch 1 and continues improving to 0.878. Right: Dev accuracy across all tasks compared against required baselines; all baselines are met or exceeded.

A.2 Generated Sonnet Examples

Figure 2 shows example sonnets generated by the fine-tuned model on dev set prompts.

Generated Sonnet Examples (Top-p sampling, $\tau=1.2$, $p=0.9$)

Example 1 (Sonnet 132 – dev set)

PROMPT (given):
 Thine eyes I love, and they, as pitying me,
 Knowing thy heart torment me with disdain,
 Have put on black, and loving mourners be,

GENERATED (model output):
 Drowned in their sweet glory to love quicken thee.
 Abomination will prostrate in thee gluttonous
 Grouching thereost thy vile delicacy,
 And doth prevent as yet thine vital part repose;
 Mine youth such lustiness will, though belated, remain.
 Even thy sins was ever present in spirit;
 Thou shalt not see upstaged that these you are come.
 Which boast can prove such as thou best feed for thy hurt

Example 2 (Sonnet 134 – dev set)

PROMPT (given):
 So, now I have confessed that he is thine
 And I myself am mortgaged to thy will,

GENERATED (model output):
 Myself I'll forfeit, so that other mine may give
 To my woe, for after thy case went me abroad.
 Thy greater luck with fools so rich
 Why sorrow of mine mother here if she have given me back those lost
 Of worms which mine point down swear by thee,
 That if mother in thy malice in thy audacity doth dreary,
 Still thence in this adulterous sepulchre it can not desire 'gainst none.
 At myself time believe it will bide,
 but shoot thyself which dwells abroad may I make leave.

Figure 2: Example sonnets generated by the fine-tuned model on dev set prompts. Given the first three lines (prompt), the model generates the remaining lines using top- p sampling ($\tau = 1.2$, $p = 0.9$). The model captures Shakespearean vocabulary and basic structure but occasionally loses coherence over longer sequences.

A.3 Per-Epoch Paraphrase Detection Results

Epoch	Train Loss	Dev Acc	Δ Dev Acc
0	0.418	0.849	—
1	0.330	0.863	+0.014
2	0.296	0.871	+0.008
3	0.271	0.874	+0.003
4	0.250	0.878	+0.004

Table 5: Per-epoch training loss and dev accuracy for paraphrase detection with AdamW.