

Efficient Fine-Tuning and Alignment of GPT-2 for Downstream NLP Tasks

Stanford CS224N Default Project

Adam Alhousiki

Department of Electrical Engineering
Stanford University
adamco27@stanford.edu

Tommy Leong

Department of Symbolic Systems
Stanford University
tommyl@stanford.edu

Kamal Elmallah

Department of Computer Science
Stanford University
kamale@stanford.edu

Abstract

We implement GPT-2 from scratch and explore parameter-efficient fine-tuning, efficient attention, and robust regularization for downstream NLP tasks. Our implementation achieves strong results on sentiment classification (0.521 dev on SST, 0.980 on CFIMDB) and paraphrase detection (0.894 dev, 0.872 test on Quora). We implement four extensions: (1) LoRA, reducing trainable parameters to 0.47% while retaining 95.4% accuracy and training $1.89\times$ faster; (2) a custom FlashAttention-2 Triton kernel achieving $1.74\times$ speedup on long sequences; (3) the SOAP optimizer; and (4) SMART regularization for sonnet generation. Our analysis reveals that FlashAttention’s benefits are workload-dependent—substantial on long sequences (1024 tokens) but counterproductive on short ones (~ 70 tokens)—highlighting the importance of matching techniques to task characteristics.

1 Key Information

- Mentor: Shicheng Liu
- External Collaborators: No
- Sharing project: No
- Late Days:

Team member	Late days remaining
Tommy Leong	4
Kamal Elmallah	3
Adam Alhousiki	2
Total pooled	6
Team size	3
Late days applied for the report	3

2 Introduction

Large language models built on the Transformer architecture have demonstrated remarkable capabilities across NLP tasks. GPT-2 (Radford et al., 2019), a decoder-only Transformer trained on next-token prediction, showed that scaling leads to emergent zero-shot and few-shot abilities. However, adapting these models to downstream tasks presents challenges in compute efficiency, memory usage, and training stability.

We implement GPT-2 from scratch and explore four axes of improvement: (1) *parameter efficiency* via LoRA (1), which injects trainable low-rank matrices into frozen weights; (2) *computational efficiency* via a custom FlashAttention-2 (2) Triton kernel; (3) *optimization* via the SOAP optimizer (Vyas et al., 2024); and (4) *robust fine-tuning* via SMART (3), combining smoothness-inducing regularization with Bregman proximal point optimization. We evaluate on sentiment analysis (SST, CFIMDB), paraphrase detection (Quora), and sonnet generation (Shakespeare).

3 Related Work

Parameter-Efficient Fine-Tuning. As models have grown to hundreds of billions of parameters (Brown et al., 2020), full fine-tuning has become prohibitively expensive. Adapter layers (Houlsby et al., 2019) add inference latency; prefix tuning (Li & Liang, 2021) reduces context length. LoRA (1) injects low-rank updates with no inference overhead, motivated by the low intrinsic dimensionality of fine-tuning updates (Aghajanyan et al., 2021).

Efficient Attention. Standard self-attention requires $O(n^2)$ memory. FlashAttention (2) tiles computation to exploit the GPU memory hierarchy, and FlashAttention-2 (Dao, 2023) improves parallelism further. Triton (Tillet et al., 2019) enables Python-based fused GPU kernels without raw CUDA.

Robust Fine-Tuning. SMART (3) combines smoothness-inducing adversarial regularization with Bregman proximal point optimization to prevent overfitting on low-resource tasks, achieving strong results on GLUE, SNLI, and ANLI.

4 Approach

Base GPT-2. We implemented GPT-2 from scratch: token/positional embeddings, masked multi-head causal self-attention with $\frac{QK^T}{\sqrt{d_k}}$ scaling, pre-LayerNorm residual connections, and weight-tied output projections (logits = hW_E^T). We verified correctness against HuggingFace and implemented AdamW (Loshchilov & Hutter, 2019) with bias correction (Kingma & Ba, 2015).

Sentiment Classification. We extract the last non-padding token’s hidden state, apply dropout ($p = 0.3$), and project to class logits. We evaluate *last-linear-layer* (frozen) and *full-model* fine-tuning.

Cloze-Style Paraphrase Detection. We formulate paraphrase detection as next-token prediction using the prompt: Question 1: "{s1}" Question 2: "{s2}" Are these questions asking the same thing? and train to predict "yes" or "no". This leverages GPT-2's pretrained language modeling rather than training a classifier from scratch.

LoRA. For weight $W_0 \in \mathbb{R}^{d \times k}$, LoRA adds $\Delta W = BA$ with $B \in \mathbb{R}^{d \times r}$, $A \in \mathbb{R}^{r \times k}$: $h = W_0x + \frac{\alpha}{r}BAx$. We inject LoRA into query and value projections of all 12 layers. With $r = 16$, this yields 590K trainable parameters (0.47% of 124M). B is zero-initialized to ensure zero initial contribution.

FlashAttention-2 in Triton. Our custom kernel avoids materializing the $N \times N$ attention matrix. It loads Q tiles into SRAM, iterates over K/V tiles computing QK^T blocks, applies causal masking, and maintains running softmax statistics via the online softmax trick:

$$m_i^{\text{new}} = \max(m_i, \text{rowmax}(S_{ij})), \quad l_i^{\text{new}} = l_i \cdot e^{m_i - m_i^{\text{new}}} + \text{rowsum}(e^{S_{ij} - m_i^{\text{new}}}) \quad (1)$$

$$O_i \leftarrow O_i \cdot e^{m_i - m_i^{\text{new}}} + e^{S_{ij} - m_i^{\text{new}}} V_j, \quad O = O/l_i \quad (2)$$

This reduces memory from $O(N^2)$ to $O(N)$. Each thread block processes one Q tile per (batch, head) pair.

SOAP Optimizer. SOAP (Vyas et al., 2024) combines Shampoo-style curvature estimation with Adam. It maintains second-order statistics $S_t^{(1)} = \beta_s S_{t-1}^{(1)} + (1 - \beta_s) G_t G_t^\top$ and $S_t^{(2)}$ analogously, rotates gradients into the eigenbasis ($\tilde{G}_t = Q_1^\top G_t Q_2$), applies Adam updates in the rotated space, then maps back: $\theta_{t+1} = \theta_t - \eta Q_1 U_t Q_2^\top$.

SMART Fine-Tuning. SMART (3) regularizes by penalizing sensitivity to input perturbations: $F(\theta) = L(\theta) + \lambda_s R_s(\theta)$, where $R_s(\theta) = \frac{1}{n} \sum_{i=1}^n \max_{\|\tilde{x}_i - x_i\|_\infty \leq \epsilon} \ell_s(f(\tilde{x}_i; \theta), f(x_i; \theta))$. It further stabilizes optimization via Bregman proximal point: $\theta_{t+1} = \arg \min_\theta F(\theta) + \mu D_{\text{Breg}}(\theta, \theta_t)$.

5 Experiments

5.1 Data

SST (Socher et al., 2013) (8,544/1,101/2,210) for 5-class sentiment; **CFIMDB** (1,701/245/488) for binary sentiment; **Quora** (283K/40K/81K) for paraphrase detection; **Shakespeare Sonnets** (131/12/12) for generation. We additionally benchmark on **WikiText-2** (seq=1024) for FlashAttention evaluation.

5.2 Experimental Details

All experiments use GPT-2 small (124M parameters, 12 layers, 768 hidden, 12 heads) on an NVIDIA L4 GPU (24GB). Last-linear-layer: 10 epochs, lr= 10^{-3} , batch 64. Full-model sentiment: 5 epochs, lr= 10^{-5} , batch 8. Paraphrase: lr= 10^{-5} , batch 8, 3 epochs. LoRA: lr= 10^{-4} , $r=16$, $\alpha=16$. SMART: lr= 5×10^{-7} , 1 epoch, $\lambda_s = 10^{-4}$, $\mu = 0.01$, $\epsilon = 10^{-6}$, 1 PGD step.

5.3 Results

Sentiment Analysis. Table 1 summarizes results. Full-model fine-tuning significantly outperforms the frozen approach and exceeds baselines on both datasets.

Table 1: Sentiment classification dev accuracy.

Fine-tune Mode	SST (5-class)		CFIMDB (binary)	
	Ours	Baseline	Ours	Baseline
Last-linear-layer	0.465	0.462	0.878	0.861
Full-model	0.513	0.513	0.980	0.976

Paraphrase Detection and LoRA. Table 2 shows results. Full fine-tuning (3 epochs) achieves 0.894 dev / 0.872 test accuracy. LoRA ($r=16$) trains **1.89 $\times$ faster** (27 vs 51 min/epoch) with 0.47% of parameters while retaining 95.4% accuracy (Figure 1). SOAP matched Adam for full fine-tuning (0.894) but slightly underperformed with LoRA (0.818 vs 0.831), suggesting limited benefit at this scale.

Table 2: Paraphrase detection on Quora. SOAP results use the same epoch count as the Adam counterpart.

Method	Params	Epoch Time	Dev Acc	Test Acc
Full fine-tuning (1 ep)	124.4M	51 min	0.871	–
Full fine-tuning (5 ep)	124.4M	55 min	0.894	0.872
Full + SOAP (5 ep)	124.4M	51 min	0.894	–
LoRA $r=16$ (1 ep)	590K	27 min	0.831	–
LoRA + SOAP (1 ep)	590K	31 min	0.818	–

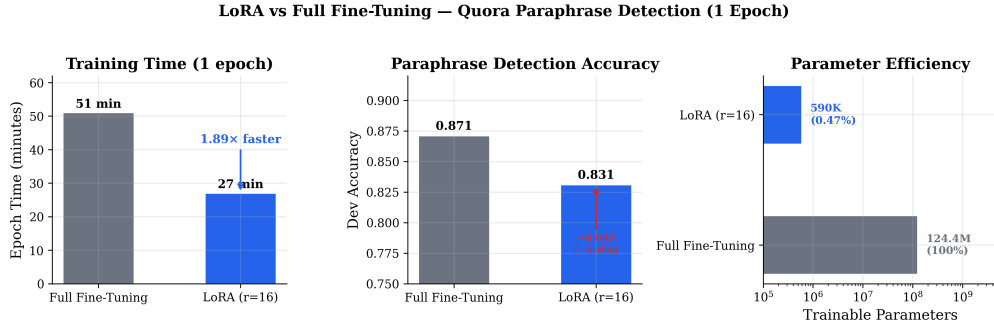


Figure 1: LoRA ($r=16$) vs full fine-tuning (1 epoch, Quora). LoRA trains 1.89 $\times$ faster with 0.47% of parameters, retaining 95.4% accuracy.

FlashAttention Benchmarks. *End-to-end* (Table 3): On WikiText-2 (seq=1024), our Triton kernel achieves **1.74 $\times$ speedup** over regular attention and **1.19 $\times$** over PyTorch SDPA, with numerically equivalent outputs (max diff < 0.002 in FP16). Forward cosine similarity is 1.000 for dQ , dK , dV .

Table 3: Attention backend benchmark on WikiText-2 (seq=1024).

Backend	Epoch Time	Speedup	Time Reduction
Regular Attention	336s	1.00 $\times$	–
PyTorch SDPA	230s	1.46 $\times$	–31.5%
Triton FlashAttn (ours)	193s	1.74 $\times$	–42.6%

Kernel microbenchmarks (Table 4, Figure 2): Isolating the kernel (fwd+bwd, batch=4, heads=12, $d=64$), naïve attention latency grows quadratically (0.95ms $\rightarrow$ 23.8ms from 128 $\rightarrow$ 1024 tokens) while Triton and SDPA scale sub-linearly. At seq=1024, Triton uses 154.6 MB vs 953.2 MB for naïve (**6.2 $\times$ reduction**). Our kernel also achieves the lowest memory among all SDPA backends (100.6 MB vs 142.6 MB for SDPA default) with competitive latency.

Table 4: Kernel microbenchmark (fwd+bwd, batch=4, heads=12, $d=64$).

Seq Len	Latency (ms)			Peak Memory (MB)		
	Triton	SDPA	Naïve	Triton	SDPA	Naïve
128	0.595	0.343	0.950	33.5	36.5	50.0
256	0.597	0.339	0.927	52.1	58.1	108.1
512	0.596	0.362	6.062	85.9	97.9	293.0
1024	0.949	0.755	23.812	154.6	178.6	953.2

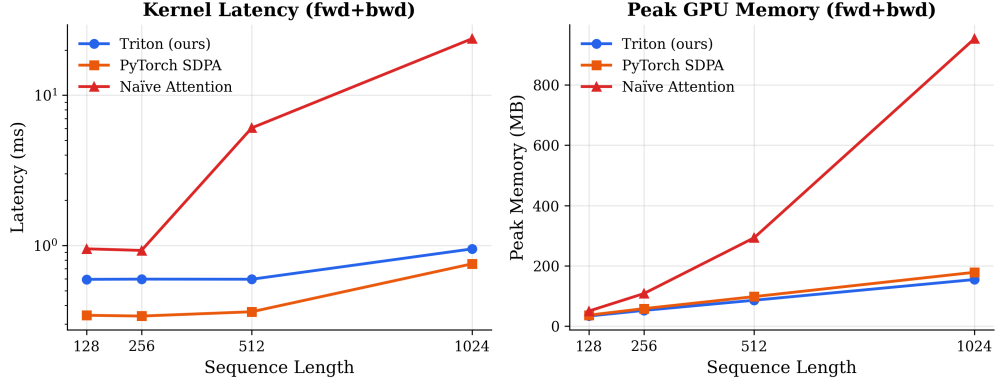


Figure 2: Kernel scaling across sequence lengths. **Left:** Naïve latency grows quadratically; Triton/SDPA stay near-constant. **Right:** Naïve memory is $O(N^2)$; tiled backends scale near-linearly.

Sequence Length Crossover. On Quora (~ 70 tokens), FlashAttention is *slower* (~ 35 min vs ~ 28 min/epoch)—kernel launch overhead dominates when sequences are too short to amortize tiling. Figure 3 visualizes this crossover.

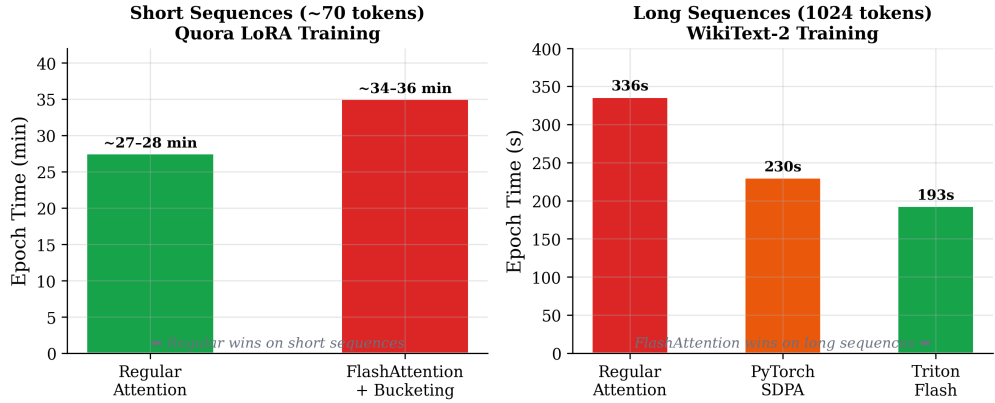


Figure 3: Sequence length crossover: FlashAttention loses on short sequences (left) but wins decisively on long ones (right).

Sonnet Generation. SMART improves dev chrF from 42.127 to **42.303** (+0.176) and achieves test chrF of **42.713**. Conservative hyperparameters ($\lambda_s = 10^{-4}$, $\mu = 0.01$) were necessary—larger values caused model drift in preliminary experiments. This is consistent with the 131-sonnet corpus being highly susceptible to overfitting.

Table 5: Sonnet generation chrF scores.

Method	Dev chrF	Test chrF
Baseline (full fine-tuning)	42.127	—
+ SMART regularization	42.303	42.713

6 Analysis

Why Triton outperforms SDPA end-to-end. Despite both implementing FlashAttention, our Triton kernel is $1.19\times$ faster end-to-end. We attribute this to: (1) *autotuning* of block sizes and warps for the specific GPU; (2) *reduced dispatch overhead* by bypassing PyTorch’s dispatch layer; and (3) *tighter fusion* of all operations into a single kernel. Interestingly, SDPA is faster in isolated

microbenchmarks (0.711ms vs 0.950ms at seq=1024), suggesting the end-to-end advantage comes from dispatch overhead amortized across layers and from our kernel’s lower memory footprint (100.6 vs 142.6 MB) reducing allocator pressure.

Sequence length crossover. Standard attention keeps the $N \times N$ matrix small enough to fit in cache for short sequences, making the simpler kernel competitive. FlashAttention’s tiling overhead only amortizes when N is large enough for bandwidth savings to dominate—practitioners should profile rather than assume FlashAttention always helps.

LoRA efficiency–accuracy tradeoff. LoRA retains 95.4% accuracy with 0.47% of parameters and $1.89\times$ speedup. The modest accuracy gap suggests paraphrase detection benefits from capacity beyond rank 16, but the efficiency gains make LoRA compelling for budget-constrained settings. The cloze-style formulation itself is key: it leverages GPT-2’s pretrained “yes”/“no” predictions rather than training a new classifier head.

SOAP. SOAP matched Adam for full fine-tuning but slightly underperformed with LoRA (0.818 vs 0.831). The low-rank adapter matrices may not benefit from curvature estimation—the optimization landscape is already well-conditioned in the constrained subspace.

SMART for sonnet generation. The improvement is modest but consistent, reflecting the tension between regularization strength and model drift on a well-trained checkpoint. The 131-sonnet corpus makes this task particularly susceptible to overfitting, and SMART’s smoothness constraint helps generalization even with conservative hyperparameters.

7 Conclusion

We implemented GPT-2 from scratch with four extensions: LoRA for parameter efficiency (0.47% params, $1.89\times$ speedup, 95.4% accuracy retained), FlashAttention-2 in Triton ($1.74\times$ speedup on long sequences), SOAP (matched Adam, no clear advantage), and SMART (+0.176 chrF on sonnets). The central finding is that efficiency techniques are workload-dependent: FlashAttention helps on long but not short sequences, and LoRA’s tradeoff depends on the task’s intrinsic dimensionality. Matching techniques to task characteristics matters more than applying them uniformly.

Team contributions

Adam Alhousiki: Base GPT-2 implementation, AdamW, sentiment classifier, paraphrase pipeline, LoRA, FlashAttention-2 Triton kernel (fwd+bwd), GCP setup. **Tommy Leong:** SonnetGPT forward pass, sonnet generation pipeline, sentiment classifier, SMART fine-tuning (adversarial regularization + Bregman proximal point). **Kamal Elmallah:** SOAP optimizer, LoRA experimentation, GCP infrastructure, hyperparameter tuning.

References

- [1] 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.
- [2] 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*, 2022.
- [3] Haoming Jiang, Pengcheng He, Weizhu Chen, Xiaodong Liu, Jianfeng Gao, and Tuo Zhao. SMART: Robust and efficient fine-tuning for pre-trained natural language models through principled regularized optimization. *arXiv preprint arXiv:1911.03437*, 2019.