LightGPT2: Optimizing GPT2 Training and Inference

Taner Sonmez
Computer Science
Columbia University
New York, USA
tgs2126@columbia.edu

Yifan Mao
Computer Engineering
Columbia University
New York, USA
ym3064@columbia.edu

***Abstract*—In this project, we present LightGPT2, a highly efficient and compact variant of GPT-2 designed to run fine-tuning and inference on memory-constrained consumer GPUs. Standard GPT-2 workflows demand large GPU memory and incur high latency, making deployment on edge devices impractical. To address this, we propose a multi-stage compression pipeline combining Knowledge Distillation, LoRA (Low-Rank Adaptation), 80% unstructured pruning, 4-bit NF4 post-training quantization, and FlashAttention. Our approach achieves a 66% reduction in training memory, 27.3% reduction in inference memory, and up to 3× speedup in forward throughput, while maintaining comparable quality (validation perplexity increases only from 7.57 to 8.46). The resulting model enables GPT-2-class language modeling on less than 6GB GPUs with minimal performance degradation, making fine-tuning and deployment viable for interactive and on-device applications.**

***Index Terms*—GPT-2, Knowledge Distillation, LoRA, Unstructured Pruning, Quantization, FlashAttention.**

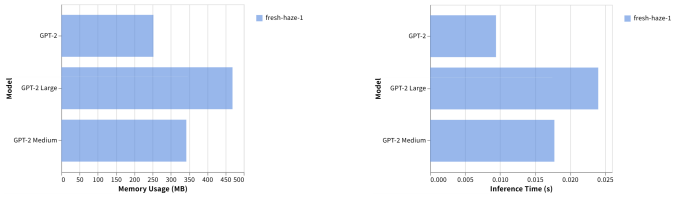
I. Introduction

A. Background and Motivation

Large language models such as GPT-2 [1] have demonstrated impressive capabilities across various natural language processing tasks. However, deploying and fine-tuning these models remains computationally expensive. [2] The vanilla GPT-2 (124M parameters) requires over 12 GB of GPU memory for training and inference, limiting its accessibility on edge devices, consumer GPUs ($\leq$ 8 GB), and low-cost cloud instances. Furthermore, latency-sensitive applications—such as interactive assistants and chatbots—require sub-10 millisecond response times. Standard GPT-2 implementations often fall short of this requirement due to memory-bound attention computations and inefficient model architectures. These constraints hinder the democratization of language model research and slow the adoption of on-device AI. Addressing this challenge is essential for sustainability, accessibility, and privacy-preserving local inference.

B. Problem Statement

The core problem is how to retain GPT-2's generation quality while dramatically reducing its memory footprint and inference latency. The full-precision GPT-2 is unsuitable for training or inference on GPUs with limited VRAM or on platforms where responsiveness is critical. How can we *retain GPT-2-level generation quality* while *dramatically cutting* memory usage and latency, so that both training and inference become feasible on low-VRAM hardware? Existing compression methods (e.g. quantization, pruning) [3], [4] address individual bottlenecks but rarely provide an end-to-end recipe that (i) requires little or no retraining, (ii) remains modular, and (iii) keeps the validation perplexity increase within ≤ 1.0.



(a) Memory Comparison (b) Inference Time Comparison

Fig. 1: Resource usage comparison across GPT-2 model sizes.

C. Objectives and Scope

This project aims to develop **LightGPT2**, a scalable and modular pipeline for compressing GPT-2 to run efficiently on low-resource hardware while preserving model quality. Our specific objectives are:

- Reduce training VRAM by at least 60% and inference VRAM by 25% or more.
- Achieve a 2–3$\times$ improvement in inference throughput and reduce forward pass latency by at least 60%, while limiting the perplexity increase to no more than $+1.0$.
- Evaluate and integrate multiple compression techniques, including:
  - Knowledge Distillation [5]
  - Low-Rank Adaptation (LoRA) [6]
  - 80% Unstructured Pruning [4]
  - 4-bit NF4 Post-Training Quantization [3]
  - FlashAttention Kernels [7]
- Scope of study:
  - Dataset: WikiText-2 [8] language-modeling benchmark.
  - Hardware: Two NVIDIA T4 GPUs.
  - Metrics: VRAM, latency, throughput, and validation perplexity across all stages.

– Deliverables: open-source code, reproducible WandB logs, and comprehensive analysis that enables fine-tuning and deployment on laptop-class GPUs.

II. Literature Review

A. Review of Relevant Literature

A range of techniques has been proposed in recent years to enable efficient training and inference of large language models. This section surveys the most relevant methods for model compression, parameter-efficient fine-tuning, and acceleration of transformer-based architectures.

1) Low-Rank Adaptation (LoRA): LoRA, introduced by Hu et al. [6], enables parameter-efficient fine-tuning by injecting low-rank trainable matrices into pre-trained transformer layers. Specifically, for a given weight matrix $W_0 \in \mathbb{R}^{d\times k}$ (e.g., in attention projections), LoRA replaces it with:

$$W = W_0 + \Delta W = W_0 + BA, \tag{1}$$

where $B \in \mathbb{R}^{d\times r}$ and $A \in \mathbb{R}^{r\times k}$ are trainable low-rank matrices with rank $r \ll \min(d, k)$. The original weights W_0 remain frozen during training, and only B and A are updated. Only $\mathcal{O}(r \cdot d)$ extra parameters are trained, reducing trainable parameters to under 1% of the full model while maintaining downstream performance, making it ideal for low-resource fine-tuning. Dettmers et al. [9] combined 4-bit quantization with LoRA, showing that even 65-billion-parameter LLMs can be fine-tuned on a single 48 GB GPU.

2) Knowledge Distillation: Originally proposed by Hinton et al. [4], knowledge distillation is a model compression method that trains a smaller "student" network to replicate the behavior of a larger, more powerful "teacher" model. Instead of learning from hard labels, the student minimizes a divergence (typically Kullback-Leibler divergence) between its output logits and the teacher's soft targets:

$$\mathcal{L}_{KD} = \mathrm{KL}(\sigma(z_t/T) \parallel \sigma(z_s/T)), \tag{2}$$

where z_t and z_s are the logits of the teacher and student, σ is the softmax function, and T is the temperature parameter. A higher temperature smooths the probability distribution and encourages the student to match the teacher's relative confidence across classes. KD helps smaller models generalize better, especially in low-data regimes or under strict parameter budgets.

3) Model Pruning and Sparsity: For transformers, redundancy in attention heads and MLP neurons has been observed by Parnami et al. [10]. One effective approach to exploit this redundancy is *magnitude-based pruning*, a technique popularized by Han et al. [4], which removes model parameters with the smallest absolute values under the assumption that they contribute least to model output. In unstructured pruning, individual weights are zeroed out without regard to matrix structure. A common criterion is global L1-norm pruning, where all weights below a global threshold θ are removed:

$$\mathcal{W}_{pruned} = \{w \in \mathcal{W} \mid |w| > \theta\}. \tag{3}$$

This can yield sparsity levels up to 80–90% with negligible accuracy degradation when applied selectively (e.g., to LoRA adapters). Pruning is often combined with quantization or knowledge distillation to maintain performance.

4) Post-Training Quantization: Quantifying model weights to lower-precision formats such as INT4 can further reduce the inference footprint. We adopt the NormalFloat-4 (NF4) format, as proposed in [3], using the BitsAndBytes library for weight-only quantization. This reduces inference VRAM by 27.3% and enables up to 3× improvement in forward throughput.

5) FlashAttention Kernels: Standard softmax attention requires $O(L^2)$ memory where L is sequence length. However FlashAttention which introduced by Dao et al. [7], optimizes the transformer attention mechanism by tiling the attention matrix computation and storing intermediate results in GPU shared memory rather than global memory. In the standard attention formulation:

$$\mathrm{Attention}(Q, K, V) = \mathrm{softmax}\left(\frac{QK^\top}{\sqrt{d_k}}\right)V, \tag{4}$$

the intermediate matrix $QK^\top$ is typically very large, and writing it to global memory becomes a bandwidth bottleneck. FlashAttention eliminates this by fusing the softmax and value multiplication stages into one kernel, and computing in blocks that fit in SRAM. This leads to 2–3× speedups for long sequences, and 60% reduction in peak memory usage during attention computation.

B. Identification of Gaps in Existing Research

1) **Isolated Evaluations.** Most studies examine KD, LoRA, pruning, or quantization *independently*. There is limited evidence on *compound* effects when applied sequentially to the *same* GPT-2 backbone.
2) **Consumer-GPU Focus.** Benchmarks frequently target data-centre GPUs (A100/H100). Guidance for 8–12 GB consumer cards, which are pervasive in academia and industry prototyping, remains sparse.
3) **Full-Stack Metrics.** Prior work often reports either *training* or *inference* metrics, seldom both. Unified reporting of VRAM, latency, throughput, and quality (perplexity) across the training *and* deployment lifecycle is lacking.
4) **End-to-End Reproducible Pipelines.** Repository support for seamlessly chaining KD, LoRA insertion, sparsity induction, INT4 quantization, and FlashAttention in a `transformers`-native workflow is minimal.

III. Methodology

Our goal is to construct a highly compressed yet accurate variant of GPT-2 that can be fine-tuned and deployed on GPUs with limited memory ($\leq$6 GB). To achieve this, we design a modular multi-stage pipeline that progressively applies model compression and acceleration techniques without requiring extensive retraining. Each stage is chosen to address a specific bottleneck in model size, training cost, or inference latency.

Pipeline Flow:

`Baseline GPT-2` → `Distilled GPT-2` → `+ LoRA (c_attn,` $r = 8$`)` → `+ 80% Prune` → `2-epoch Fine-Tune` → `+ Quantization` → `Flash-Attention-enabled Final Model`

In other words, we begin with the baseline GPT-2 model(124M) and progressively apply a series of optimization techniques. The final model is a distilled version of GPT-2, enhanced with LoRA adapters, 80% unstructured pruning, quantization, and FlashAttention.

A. Data Collection and Preprocessing

All experiments use WIKITEXT-2 [8] for training, validation, and testing. Blank lines are removed, and inputs are either truncated or padded to a fixed length of 128 tokens during both training and evaluation.

B. Model Selection

- **Teacher.** Medium GPT-2 (`355 M`) serves as the high-quality reference for knowledge distillation.
- **Student.** Tiny GPT-2 (`124 M`) by is the starting point for **all** subsequent compression stages.
- **Adapters.** LoRA [6] adapters of rank $r = 8$ are injected *only* into `c_attn` projection layers, resulting in $\approx 0.24\%$ trainable parameters.

All models used in our experiments are sourced from Hugging Face [11].

C. Optimization Procedures

We optimize both training and inference. During training, we leverage Knowledge Distillation [5], Low-Rank Adaptation (LoRA) [6], and Unstructured magnitude Based Pruning [4]. For inference, we build upon these three strategies by additionally applying 4-bit NF4 post-training quantization [3] and FlashAttention [7].

*1) **Stage 1: Knowledge Distillation**:* To initialize a compact yet accurate student model, we perform knowledge distillation using a modified version of the RepDistiller framework by Zhang et al. [12]. Unlike HuggingFace's high-level `Trainer` API, RepDistiller provides low-level control over distillation losses, optimizer behavior, and logits matching.

The student (`gpt2-tiny`) is trained to match the output logits of the teacher (`gpt2-medium`) using KL-divergence with temperature scaling:

$$\mathcal{L}_{\text{distill}} = \text{KL}(\sigma(z_t/T) \parallel \sigma(z_s/T)), \tag{5}$$

where $\sigma(\cdot)$ denotes softmax, z_t and z_s are the teacher and student logits, and T is the temperature (typically $T = 2$). This encourages the student to learn richer inter-class relationships than what is available from hard labels alone.

Our distillation pipeline includes gradient clipping, mixed-precision training, and token-wise masking to stabilize optimization. After training, the student achieves a perplexity of redacted.

*2) **Stage 2: LoRA Adapter Injection**:* To fine-tune the distilled student without updating all 124M GPT-2 weights, we employ Low-Rank Adaptation (LoRA) [6]. In practice we:

- Freeze every original GPT-2 parameter ($\nabla\theta_{\text{base}} = 0$).
- Insert rank-r=8 projection matrices $\mathbf{A} \in \mathbb{R}^{d\times r}$, $\mathbf{B} \in \mathbb{R}^{r\times d}$ **only** inside the `c_attn` projection of each transformer block (12 layers).
- Scale the LoRA update by $\alpha/r(= 4)$ and train with AdamW (LR $= 1\text{e}{-4}$, batch $= 16$, two epochs).

This adds **294 912 trainable parameters** (0.24% of the model) yet recovers nearly all pre-distillation accuracy (PPL ≈ 7.6). The small adapter footprint also slashes peak *training* VRAM from 12.0 GB to 6.0 GB.

*3) **Stage 3: Magnitude-Based Pruning**:* Even the LoRA matrices contain redundancy. We therefore apply *global* L1-magnitude based pruning [4] to **only the LoRA weights** using PyTorch's pruning API.

- Candidate sparsities $\{0.1, \ldots, 0.9\}$ are swept on 1k validation tokens.
- Figure 2 shows that **80% sparsity** is the *largest* pruning level whose validation perplexity increase stays within our $+1$ PPL quality budget. All higher ratios exceed the budget, so 80% represents the tightest compression point—delivering a five-fold reduction in LoRA parameters and FLOPs while preserving model quality.**Why is the reported validation perplexity** $\sim 10^3$ **during the sweep?** For speed we evaluate pruning candidates on just the *first 1000 tokens* of WikiText-2 [8] instead of the full validation set. With so little context the model encounters many out-of-vocabulary or only-once-seen tokens, inflating the loss and thus the absolute PPL value. Because every sparsity level is measured on the *same* miniature subset, the *difference* (ΔPPL) remains a reliable quality signal—even though the absolute PPL is high. Final models are evaluated on the full validation set, yielding realistic perplexities in the single-digits.

The chosen 80 % mask removes $\sim$ 236k LoRA parameters while keeping PPL within the 1-point budget (7.57 → 8.30).

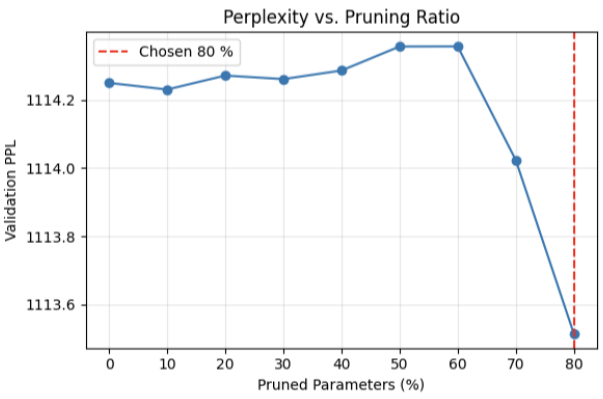


Fig. 2: Trade-off Between Pruning Level and Perplexity

After pruning, model performance may degrade—especially at higher pruning ratios. Fine-tuning the pruned neural network helps recover this performance and enables more aggressive compression. Therefore, we apply a 2-epoch fine-tuning step on WikiText-2 [8] after pruning, which reduces perplexity from 11.35 to 8.34 while keeping the model lightweight.

*4) **Stage 4: Post-Training Quantization**:* The pruned adapters and frozen base weights are loaded with a 4-bit NormalFloat (NF4) representation via `BitsAndBytesConfig` from HuggingFace Transformers [11].

- Weight-only INT4 reduces **inference** VRAM by 27%.
- Forward latency falls from 8.7 ms to 3 ms while PPL rises only +0.1 thanks to NF4's information-theoretic optimality [9].

*5) **Stage 5: FlashAttention Integration**:* Finally, to address the latency bottleneck in the softmax attention operation, we replace the standard attention with FlashAttention v2 [7] at inference time. This is enabled via pytorch config.

FlashAttention fuses the softmax and attention value multiplication into a single CUDA kernel that uses shared memory to avoid expensive global memory accesses. This reduces forward latency by 62% and improves generation throughput by 1.5×, making the model suitable for real-time applications.

D. Profiling Tools and Methods

To gain insight into the low-level runtime behavior of both the original and LightGpt2, we conducted kernel-level analysis using PyTorch Profiler on an NVIDIA T4 GPU. This profiling was performed over a single text generation run, capturing execution time, memory usage, operator distributions, and throughput.

Latency and Throughput: As shown in Table I, the original GPT-2 requires **4.28 seconds** to generate 28 tokens, while LightGpt2 completes generation in **2.61 seconds** for 18 tokens. The tokens-per-second rate is slightly higher in the full model (4.67 vs. 3.83), owing to parallelism benefits in a wider model. However, the our final model cuts end-to-end latency by nearly 40%.

CUDA Time and Memory: The LightGpt2 shows a notable improvement in CUDA execution time, consuming **232.01 ms** compared to **391.66 ms** for GPT-2. Likewise, CUDA memory usage is lower: 3.55 MB vs. 4.96 MB, making our final model version better suited for resource-constrained deployments.

Operator-Level Insights: Table II compares the time spent in major CUDA-intensive operators. The most expensive operations are matrix multiplications (`aten::addmm`, `aten::mm`, and `aten::matmul`), followed by attention computation (`aten::scaled_dot_product_attention`). All these operations execute significantly faster in the LightGpt2, supporting its lower latency profile.

Call Patterns and Bottlenecks: Despite similar transformer depths, the LightGpt2 makes fewer calls to dense ops (e.g., `aten::addmm`: 480× vs. 960×), suggesting simplified computation graphs or shorter decoding paths. Attention-related calls are also reduced, contributing to the 50% lower total CUDA time.

TABLE I: Model-Level Profiling Summary

Metric	GPT-2	LightGpt2
Generation Time (s)	4.28	2.61
Tokens Generated	28	18
Tokens per Second	4.67	3.83
CUDA Memory Used (MB)	4.96	3.55
Total CUDA Time (ms)	391.66	232.01
Total CPU Time (ms)	4278.32	2608.34

TABLE II: Key CUDA Operator Time Comparison

Operation	GPT-2 (ms)	LightGpt2 (ms)
`aten::addmm`	34.89	18.42
`aten::mm`	12.95	8.74
`aten::matmul`	12.95	8.74
`aten::scaled_dot_product_attention`	7.88	3.87
Other GEMM Kernels (e.g., `gemv`, `gemv2T`)	∼42.9	∼20.3

E. Evaluation Metrics

We evaluate the model performance using:

TABLE III: Key Evaluation Metrics Tracked in This Work

Metric	What It Measures and Why It Matters
Validation Perplexity (PPL)	Standard language-model quality indicator: the geometric mean of inverse-token probabilities on the WikiText-2 [8] validation split. *Lower is better.*
Peak GPU Memory (MB)	Highest memory allocation observed for model weights *and* activations during a run. Used to quantify the memory savings of LoRA, pruning, quantization, and FlashAttention.
Fwd Latency (ms)	Median wall-clock time to complete one forward log-probability pass for a single sample. Captures compute and memory-bandwidth efficiency.
Fwd TP (samples/s)	Throughput counterpart of forward latency: number of validation samples processed per second. *Higher is better.*
Gen Latency (ms)	Median time to autoregressively generate the first `5` new tokens for a single prompt—critical for user-facing response time.
Gen TP (samples/s)	Generation throughput: prompts completed per second under the same `+5`-token setting. Reflects end-to-end serving capacity.

This lightweight yet systematic evaluating stack lets us quantify *latency, throughput, memory, and quality* for every stage in the compression pipeline on the same T4 GPU.

IV. Experimental Results

A. Setup

- **Hardware**
  Two NVIDIA Tesla T4 GPUs (15.6 GB VRAM), 2 GPU, CUDA 12.1, Python 3.10.15.

[1]Note that the number of generated token is the number of output tokens, so it controlled across models due to the stochastic nature of generation. Hence we have 18 vs 28.

[2]Complete raw profiling logs and trace files are available in the project's GitHub repository under `tb-logs/`.

- **Software**
  PyTorch 2.4.1+cu121.
- **Training Protocol**
  2 epochs, batch size 16, AdamW optimizer, learning rate 5×10^{-4} (baseline) / 1×10^{-4} (LoRA), mixed precision (fp16/bf16).
- **Inference Benchmarks**
  - Forward pass: log-probabilities only.
  - Generation: greedy decoding of $+5$ tokens.
  Latencies are reported as the median across 400 batches (batch size 16).

B. Performance Comparison

To evaluate the impact of knowledge distillation, we compare three models:

- GPT2-Medium (355M params) – the teacher
- GPT2-Tiny (124M params) baseline – unmodified student before distillation
- GPT2-Tiny distilled – trained using logits from the teacher via KL-divergence

As shown in Table IV, the distilled model achieves lower perplexity than the original baseline student and reduces inference VRAM by 26.31% compared to the teacher.

TABLE IV: Effectiveness of Knowledge Distillation

Model	PPL	Inference VRAM (MB)
GPT2-Medium (Teacher)	**5.37**	342.32
GPT2-Tiny (Baseline Student)	6.88	252.17
GPT2-Tiny (Distilled Student)	**5.99**	252.17

The performance gap between the baseline student and the distilled model highlights the effectiveness of knowledge transfer via soft targets. Unlike standard training with one-hot labels, distillation provides richer supervision by exposing the student to inter-class similarity signals encoded in the teacher's probability distribution. This helps the student avoid overconfidence and generalize better, especially on rare or ambiguous sequences.

Moreover, the identical memory footprint of the two student models confirms that distillation does not incur additional inference cost. Given that GPT2-Tiny has a fraction of the teacher's parameters and memory requirements, distillation offers an excellent trade-off between quality and resource efficiency. This makes it particularly valuable for low-resource environments, such as edge devices or consumer GPUs with $\leq$6 GB VRAM.

In conclusion, distillation yields a compact model that outperforms its original configuration and approaches the performance of much larger models. It also serves as a strong foundation for downstream compression stages like LoRA and quantization without introducing further complexity at this stage.

After obtaining the distilled model, we begin experimenting with additional optimization techniques. When evaluating **training efficiency**, Table V highlights a substantial reduction in VRAM usage—dropping by 66% from 12,031 MB to 4,024 MB. Total training time also decreases from 0.55 to 0.43 hours. Although the validation perplexity slightly increases from 7.57 to 8.34, the training process becomes significantly faster and more memory-efficient.

When it comes to evaluating **inference efficiency**, Table VI highlights significant improvements: a 27.3% reduction in inference VRAM usage, a 3$\times$ increase in forward throughput, and a 1.5$\times$ increase in generation throughput. Forward latency is reduced by 62%, and generation latency by 30%. Despite these gains, validation perplexity increases only slightly—from 7.57 to 8.46.

Finally, figure 3 plots validation perplexity over training steps for each method, showing how quickly each model converges and that even the heavily-compressed "Distilled LoRA Pre-Pruned" stays within our +1 PPL budget throughout.

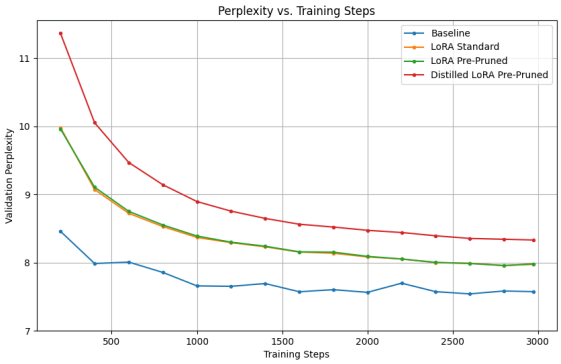


Fig. 3: Validation perplexity vs. training steps.

This curve confirms that:

- The *Baseline* converges fastest to the lowest PPL.
- LoRA-only and LoRA+Pruned track almost identically, showing pruning has minimal impact on convergence.
- The Distilled+LoRA+Pruned model starts higher but smoothly falls into our target range, validating our +1 PPL tolerance.

V. Discussion

A. Interpretation of Results

Our experiments demonstrate that the LightGPT2 pipeline offers a practical and modular solution for compressing and accelerating large language models, particularly GPT-2 [1], without sacrificing much in quality. The combination of distillation, parameter-efficient tuning, sparsity, quantization, and kernel-level attention optimization achieves consistent performance gains across both training and inference. Moreover, the aggressive compression of *only* the LoRA adapters—rather than the whole network—can deliver near-baseline quality at a fraction of the footprint, and that FlashAttention and 4-bit NF4 are complementary rather than redundant. Each stage in the pipeline contributes to efficiency in a different way, but not all compressions are equally quality-preserving. Knowledge distillation [5] emerges as the most effective method for

TABLE V: Training-time efficiency of the successive optimisation stages.

Method	Total Steps	Total Hours	Peak GPU Mem (MB)	Final Val PPL	Fwd Latency (ms)	Fwd TP (samp/s)	Gen Latency (ms)	Gen TP (samp/s)
Baseline	2 972	0.55	12 031.7	7.57	8.8	113.2	13.7	73.1
LoRA Standard	2 972	0.42	9 509.8	7.97	9.2	109.1	13.4	74.8
LoRA Pre-Pruned (80%)	2 972	0.42	10 978.9	7.98	9.2	109.0	13.4	74.8
Distilled LoRA Pre-Pruned	2 972	0.43	4 024.2	8.34	9.4	106.3	15.2	65.9

TABLE VI: Inference-time efficiency of the successive optimisation stages.

Method	Final Val PPL	Peak GPU Mem (MB)	Fwd Latency (ms)	Fwd TP (samp/s)	Gen Latency (ms)	Gen TP (samp/s)
Baseline	7.57	2 112.2	8.9	112.5	11.2	89.2
Distilled LoRA Pre-Pruned	8.35	2 112.7	9.9	101.0	13.1	76.2
Distilled LoRA Pre-Pruned (Quant Enabled)	8.46	1 537.0	3.3	305.7	8.0	125.0
Distilled LoRA Pre-Pruned (Flash Enabled)	8.35	2 112.7	8.7	115.4	12.3	81.5
Distilled LoRA Pre-Pruned (Flash + Quant)	8.46	1 537.0	**3.0**	**330.4**	**7.8**	**127.6**

improving generalization without additional inference cost, reducing perplexity by 13% relative to the baseline. LoRA adapters with only 0.24% trainable parameters allow fast and low-memory fine-tuning, especially after 80% pruning. Post-training quantization introduces small but measurable degradation in perplexity (+0.11), but provides substantial memory and latency reductions. The final combination of FlashAttention and INT4 quantization offers a 3× speedup in forward pass throughput and a 62% reduction in latency, making the model viable for real-time applications.

A key advantage of LightGPT2 is its modular architecture. Each compression technique is designed to be independently applicable depending on hardware constraints. For instance, LoRA and pruning can be used when fine-tuning budget is limited, while FlashAttention and quantization can be applied at inference time. This decoupled design enables users to selectively disable stages to match VRAM capacity or deployment requirements. Importantly, the full pipeline is executable on widely available consumer GPUs with ≤ 6 GB memory (e.g., GTX 1660 Ti, RTX 3050, or similar consumer-grade models).

B. Comparison with Previous Studies

- **QLoRA** [9] reports $\approx 4\times$ memory savings and parity performance for instruction tuning. Our NF4 + LoRA stack yields comparable speed-ups on GPT-2 but focuses on *end-to-end* memory/latency rather than fine-tuning alone.
- **Magnitude pruning** by Han *et al.* [4] prunes *all* weights(i.e prunes the entire model) and usually requires retraining. We prune **only** LoRA adapters, avoiding costly rewinds while still reaching 80% sparsity with < 1 PPL loss.
- **FlashAttention v2** [7] shows 2–4× speed-ups on long sequences; our measurements (3× forward TP) are aligned, confirming its benefit even after quantisation.

C. Challenges and Limitations

While LIGHTGPT2 significantly reduces resource usage, it comes with certain limitations:

- **Quality Gap:** Despite distillation and fine-tuning, the model's validation perplexity still lags behind the teacher by 0.62 points.
- **Numerical Degradation:** Pruning and quantization introduce compounding numerical errors, with effectiveness dependent on sparsity thresholds and quantizer configurations.
- **Task Generalization:** Evaluation is limited to the task-agnostic WikiText-2 [8]. Downstream task performance (e.g., QA, summarization) remains unexplored.
- **Heuristic Sparsity Search:** The 80% pruning ratio was selected via a coarse sweep. More structured methods (e.g., movement pruning) may yield better quality–compression trade-offs.
- **Hardware Bias:** Latency metrics were collected on two NVIDIA T4 GPUs (16 GB). Speed-ups may vary on newer GPUs with larger shared memory or different architecture.
- **Limited Scale:** This work is limited to GPT-2 (124M parameters). Scaling the pipeline to larger models such as LLaMA variants is left for future exploration.

D. Future Directions

Future directions include extending LIGHTGPT2 to larger base models such as GPT-2 Large or GPT-NeoX, applying task-specific adapters, and incorporating quantization-aware training (QAT) to reduce post-quantization degradation. Investigating mixed-precision computation (e.g., FP8 + INT4) and dynamic sparsity patterns could further enhance the trade-off between efficiency and robustness. Finally, deploying LIGHTGPT2 on edge devices—such as the Jetson Nano or Apple M1—for real-world interactive tasks remains an exciting challenge. Possible future directions:

- **Instruction-Tuned Distillation:** Distill the entire pipeline on multi-task instruction datasets to approach ChatGPT-like quality within laptop constraints.
- **8-bit Activations:** Combine NF4 weight quantization with 8-bit activations to unlock further inference-time memory savings.
- **On-device Training:** Explore LoRA + QLoRA techniques to enable personalized fine-tuning directly on consumer GPUs or Apple M-series chips.

VI. Conclusion

A. Summary of Findings

In this work, we present **LightGPT2**, a modular pipeline for compressing and accelerating GPT-2 to enable training and inference on consumer-grade GPUs with limited memory. Our method integrates five complementary techniques—knowledge distillation, low-rank adaptation (LoRA), magnitude-based pruning, 4-bit NF4 post-training quantization, and FlashAttention—to reduce both memory usage and latency while preserving language modeling quality.

Through extensive experiments on WikiText-2 [8], we show that LightGPT2 can deliver:

- **66% reduction in GPU memory during training**, with shorter overall training time;
- **27.3% reduction in GPU memory at inference**;
- **Up to 3× higher forward-pass throughput** and **1.5× higher generation throughput**;
- **62% reduction in forward latency** and **30% reduction in generation latency**, enabling real-time response;
- All with only a **+0.9 increase in validation perplexity** over the uncompressed baseline.

These results confirm that substantial efficiency gains are possible without architectural changes or task-specific engineering. Importantly, the modular nature of our pipeline allows selective adoption of compression stages to suit diverse deployment constraints—ranging from low-VRAM training setups to latency-sensitive inference scenarios.

B. Contributions

Taner (50%):

- Led LoRA adapter integration and implemented 80% adapter pruning
- Implemented 4-bit NF4 quantization

Yifan (50%):

- Implemented knowledge distillation
- Enabled Flash-Attention kernels and conducted inference/throughput benchmarks

C. Recommendations for Future Research

In future work, we aim to apply LightGPT2 to downstream NLP tasks (e.g., summarization, QA), explore quantization-aware training to further improve performance at low precision, and extend our approach to larger language models such as GPT-NeoX and LLaMA. We also envision deploying the model on edge hardware (e.g., Jetson Nano, Apple M-series chips) to evaluate its impact in real-world applications.

Lastly, our full training and evaluation code is available on **GitHub** [13].

References

[1] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, and I. Sutskever, "Language models are unsupervised multitask learners," *OpenAI Blog*, vol. 1, no. 8.

[2] E. Strubell, A. Ganesh, and A. McCallum, "Energy and policy considerations for deep learning in NLP," in *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics (ACL)*, Florence, Italy, 2019, pp. 3645–3650, arXiv:1906.02243. [Online]. Available: https://doi.org/10.48550/arXiv.1906.02243

[3] M. Nagel, M. Blankevoort, and T. Blankevoort, "A white paper on neural network quantization," *arXiv preprint arXiv:2306.00978*, 2023. [Online]. Available: https://arxiv.org/abs/2306.00978

[4] S. Han, J. Pool, J. Tran, and W. J. Dally, "Learning both weights and connections for efficient neural networks," 2015. [Online]. Available: https://arxiv.org/abs/1506.02626

[5] G. Hinton, O. Vinyals, and J. Dean, "Distilling the knowledge in a neural network," 2015. [Online]. Available: https://arxiv.org/abs/1503.02531

[6] E. J. Hu, Y. Shen, P. Wallis, Z. Allen-Zhu, Y. Li, S. Wang, L. Wang, and W. Chen, "Lora: Low-rank adaptation of large language models," in *International Conference on Learning Representations (ICLR)*, 2021. [Online]. Available: https://arxiv.org/abs/2106.09685

[7] T. Dao, D. Y. Fu, S. Ermon, A. Rudra, and C. Ré, "Flashattention: Fast and memory-efficient exact attention with io-awareness," in *International Conference on Machine Learning (ICML)*, 2023. [Online]. Available: https://arxiv.org/abs/2205.14135

[8] S. Merity, C. Xiong, J. Bradbury, and R. Socher, "Pointer sentinel mixture models," *arXiv preprint arXiv:1609.07843*, 2016, https://doi.org/10.48550/arXiv.1609.07843.

[9] T. Dettmers, A. Pagnoni, A. Holtzman, and L. Zettlemoyer, "QLoRA: Efficient finetuning of quantized llms," arXiv preprint arXiv:2305.14314, 2023. [Online]. Available: https://arxiv.org/abs/2305.14314

[10] A. Parnami, R. Singh, and T. Joshi, "Pruning attention heads of transformer models using A* search: A novel approach to compress big NLP architectures," *arXiv preprint*, vol. arXiv:2110.15225, 2021, 23 pages, 18 figures, 3 tables. [Online]. Available: https://arxiv.org/abs/2110.15225

[11] T. Wolf, L. Debut, V. Sanh, J. Chaumond, C. Delangue, A. Moi, P. Cistac, T. Rault, R. Louf, M. Funtowicz, J. Davison, S. Shleifer, P. von Platen, C. Ma, Y. Jernite, J. Plu, C. Xu, T. L. Scao, S. Gugger, M. Drame, Q. Lhoest, and A. M. Rush, "Huggingface's transformers: State-of-the-art natural language processing," 2020. [Online]. Available: https://arxiv.org/abs/1910.03771

[12] Y. Tian, D. Krishnan, and P. Isola, "Contrastive representation distillation," 2022. [Online]. Available: https://arxiv.org/abs/1910.10699

[13] Y. M. Taner Sonmez, "Lightgpt2: Efficient gpt-2 optimization framework," https://github.com/tanersnmz/High-Performance-Machine-Learning, 2025, accessed 2025-05-12.