Lightweight Transformers for Text Generation

Distillation, Pruning, and Quantization of BART-large and GPT-2

Six Degrees of Inner Turbulence

Name	UFID
GuangYan An	9853-4118
Xiaolin Zheng	5400-7588

Contents

1 Abstract

This report mainly presents the development and outcomes of light model optimizations on the pre-trained BART-large-CNN model on text summarization tasks and GPT-2 on text prediction tasks. By leveraging deep learning model optimization techniques such as knowledge distillation, movement pruning, quantization (post-training quantization, PTQ and quantization-aware training, QAT) on transformer-based models, we reduced 75% of disk storage for the BART-large-CNN model and 69% for the GPT-2 model. The ROUGE score for summarization retained 95% of the origin, and the perplexity score of prediction increased from 18 to 65. Additionally, inference time was reduced by 37%.

2 Introduction

Text generation systems play a critical role in improving user efficiency in applications like typing assistants, chatbots, and new digests. Traditional methods, often based on n-gram or rule-based models, struggle with understanding complex sentences and providing accurate suggestions over longer contexts. The emergence of transformer-based models has transformed Natural Language Processing (NLP). Models like BERT, BART, and GPT-2 leverage bidirectional context, enabling more accurate predictions. Transformers also benefit from large pre-trained models and extensive datasets, but they come with the drawback of requiring significant computational resources, which can limit their use in real-time applications.

Many companies, such as Microsoft, Meta, and Apple, have developed their own text completion / summarization systems. However, these solutions often have limitations, such as device or platform restrictions. Our motivation is to create a context-aware text completion / summarization system that balances quality, speed, and efficiency, working across platforms without sacrificing performance.

Regarding these challenges, we employed several established techniques to reduce the size of pre-trained models and enhance model inference speed, while keeping accuracy loss within acceptable limits. These techniques include knowledge distillation, movement pruning, and quantizations. In the following sections, we will present the development, internals, and outcomes of the optimized pre-trained models. Section 3 discusses the chosen datasets, prerequisites, and miscellaneous setups. Section 4 presents the knowledge distiller that we have implemented. Section 5 presents how we applied movement pruning techniques to the pre-trained Transformer models. Section 6 explores the experiments and performance of our PTQ and QAT implementations. Section 7 summarizes the final and overall performance, inference time, and memory footprint generated and evaluated by using our distillation-pruning-quantization pipelines on pre-trained Transformer models. Section 8 concludes this project and discusses various potential future improvement we have observed and summarized in these months during the project.

3 Experimental Setup

We used a subset of OpenWebText dataset[1], an open-source replication of OpenAI's WebText, for autoregressive text prediction and a BBC news dataset containing original posts and their corresponding summarizations for text summarization. We utilized HuggingFace's `transformers`[2] library, which provides the pre-trained definitions for BART and GPT-2 models, as well as the `Trainer` and `Seq2SeqTrainer` APIs, to fine-tune these models during training. The latter allowed us to focus more on optimizing the models rather than implementing training routines (trainer, checkpointing, etc.) from scratch. The model optimization pipelines were trained and evaluated on an A100 GPU with 40GB memory on Google Colab and required approximately 5 hours of runtime since only a small subset of the original dataset was used (We could only afford this setting). Before applying the optimizations, we first fine-tuned the pre-trained models on the prepared datasets to warm them up, also for later performance comparison.

4 Knowledge Distillation

4.1 Introduction

Knowledge Distillation[3] was proposed in 2015 as a technique for transferring knowledge from a large, complex model (teacher model) to a smaller, more efficient model (student model). The key idea is that the student model can achieve comparable performance to the teacher model by learning soft labels of the teacher: probability distributions instead of learning the true labels only. This allows the student to learn intermediate representations from the teacher. The loss function of knowledge distillation can be expressed as:

$$L_{\text{KD}} = \alpha L_{\text{CE}}(y, \hat{y}) + (1 - \alpha) T^2 L_{\text{KL}}(q_t, q_s) \tag{1}$$

In the above expression, $L_{\text{CE}}(y, \hat{y})$ represents the loss between prediction probability distribution $\hat{y}$ and true labels y using traditional cross entropy loss. $L_{\text{KL}}(q_t, q_s)$ expresses the difference between the output distributions of the student and the teacher. T is the temperature hyperparameter, used to soften softmax distributions of q_t and q_s. Finally, α controls the balance between different kinds of losses.

4.2 Implementation

BART-large-CNN[5] was pre-trained and fine-tuned on the CNN Daily Mail dataset and has a powerful ability of text summarization. It has 12 Transformer encoder layers and 12 Transformer decoder layers, the model being in total of 1.51GB. GPT-2[6] was a pre-trained autoregressive decoder model using a causal language masking objective, containing 12 Transformer decoder layers with a total size of 0.548 GB. We extracted the layers 0, 2, 5, 7, 9, 11 from the encoder / decoder to form the student's encoder / decoder, thereby halving the model's storage. Weight state dicts of these layers are also loaded to avoid pre-training from scratch.

In our implementation, we chose to use `MSE` to match the logits output at each hidden layer as described in Chen et al. (2020)[4] since this can encourage the student to even match teacher hidden states. The loss function is updated as follows, where we want the student to generate hidden layers weights as comparable to the teacher as possible:

$$L_{\text{KD}} = \alpha L_{\text{CE}}(y, \hat{y}) + \beta L_{\text{MSE}(hid_t, hid_s)} + \gamma T^2 L_{\text{KL}}(q_t, q_s) \tag{2}$$

For the α, β, γ, T hyperparameters, we set α to 0.1, β to 3, γ to 0.8, and T to 2, since we aim to maximize the hidden state learning by the student from the teacher.

As shown in Figure 1, this image illustrates an example encoder-decoder Transformer (e.g. BART) teacher model with 3 layers for each on the left. On the right a distilled student model extracts the first and last layers of the teacher's encoder and decoder (using the same colors) and forms a new encoder and decoder. Arrows in this figure show the data flow: `input embeds` are the text embedding input, which will become `Q,K,V` in encoders. Encoder's last hidden layer states will be transformed to `K,V` used in cross-attention of decoder layers, and the `labels` are fed as `Q,K,V` in self-attention but as `Q` only in cross-attention of decoder layers.

Figure 2 presents an example 3-layer decoder-only Transformer (e.g. GPT-2) teacher model on the left, which is much simpler than the previous one since decoder-only models can choose not to use cross-attention, meaning it only leverages self-attention. On the right a distilled student model extracts the first and last layers of the teacher's decoder (using same colors) and forms a new decoder. `input embeds` will become `Q,K,V` in decoders. Finally the final logits generated by the decoder will be compared with the original text embeddings using cross entropy to calculate the loss because of autoregression.

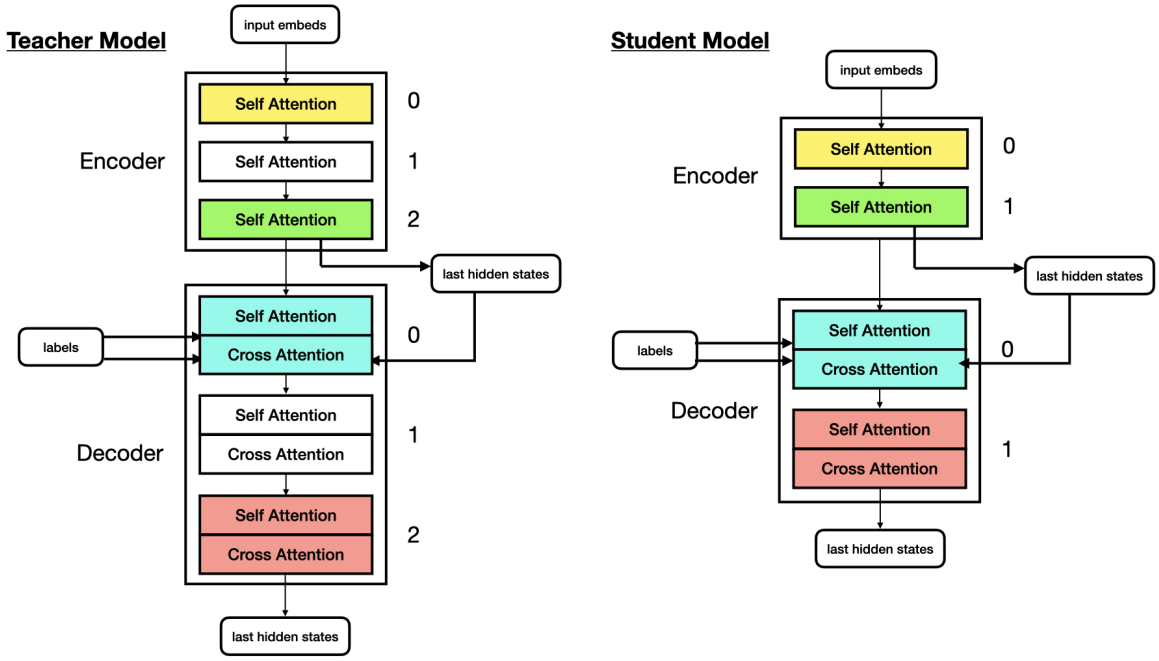


Figure 1: Example of distillation of Encoder-Decoder model, e.g. BART-large

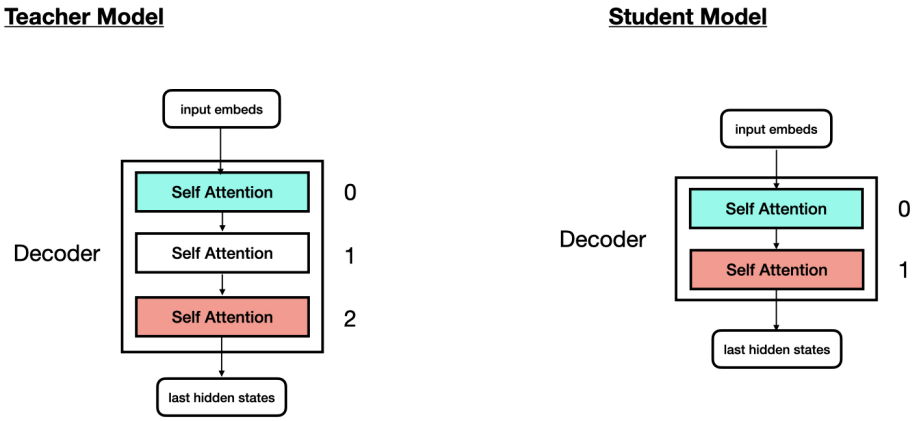


Figure 2: Example of distillation of Decoder-Only model, e.g. GPT-2

4.3 Evaluations

We have evaluated the performance, inference time, and the reduction of storage for our distiller with the settings discussed above. As the table 1 shows below, the distiller for BART-large-CNN on text summarization had a good result: not only did our student have nearly the same rouge scores as the teachers', but it also had 26.7% inference time reduction and 43% model size reduction. The rouge scores range from 0 to 1, with higher values indicating better quality. Please note that the model size reduction is less than 50% but that is normal since Transformer models have a shared embedding layer above the encoder to transform text ids to text embeddings, and another linear language modeling head layer below the decoder to map hidden states to vocabulary logits.

Name	**BART-large-CNN**	
Metric	**Teacher Model**	**Student Model**
Encoder Layers	12	6
Decoder Layers	12	6
ROUGE-1 Score	0.717	0.716
ROUGE-2 Score	0.617	0.617
ROUGE-L Score	0.541	0.551
ROUGE-Lsum Score	0.559	0.604
Inference Time	59.25s	43.43s
Model Size	1.51GB	0.86GB

Table 1: Comparison of Teacher and Student Models for BART-large-CNN

The result of distiller for GPT-2 are shown below in the table 2. Since we use GPT-2 for text prediction and it is an autoregressive decoder-only model, perplexity was used to evaluate model performance. The lower the perplexity score, the better the model's predicted probability distribution matches the true data distribution, indicating higher quality in generated text under certain conditions. The table suggests that the perplexity score downgraded from 18.25 to a "more perplexing" 31.98, which we believe it is due to the small size of the dataset, and because on text generation we did not train it to full convergence due to a lack of resources. However, the inference time shows 19% reduction, with the model reduced by 43% (the latter one is same as the result from BART-large-CNN).

Name	**GPT-2**	
Metric	**Teacher Model**	**Student Model**
Decoder Layers	12	6
Perplexity	18.25	31.98
Inference Time	43.18s	34.95s
Model Size	0.548GB	0.31GB

Table 2: Comparison of Teacher and Student Models for GPT-2

5 Movement Pruning

5.1 Introduction

Pruning is a model optimization technique aimed at reducing the size and computational complexity of deep learning models by removing less important parameters or structures. There are several different classification approaches for pruning, including by granularity (weight, neuron, channel, layer), by timing (pre-training, during training, post-training), and by target (structural, unstructural). Our pruning strategy focuses on neurons in FFN layers and attention heads in `QKV` projection layers. Specifically, we train pruning masks for both FFN and `QKV` projections separately, and finally perform pruning using the masks. Therefore, our approach can be categorized as structural movement pruning, taking neurons and attention heads as the minimum granularity.

5.2 Implementation

We assign a uniformly drawn score to each neuron in an FFN layer and to each head in an attention layer. Each time during forward passes, we use a threshold `K` to select the top `K` smallest scores and drop the corresponding neurons and attention heads by transforming the scores into masks. Regarding neurons and attention heads as pruning granularity, we should notice that once we have determined which attention head is useless, we should remove that same head from all `QKV` projections and the final output projection in the current attention. Also for FFN layers that contain two linear layers `fc1` and `fc2`, once we have decided which neuron can be pruned in `fc1`, we must remove the corresponding neuron defined in `fc2`. Therefore, we call our trained scores "shared score" or "shared mask", since they are shared among `QKV` attention projections, and among FFN layers.

During the training process in order to find the best shared masks in the Transformer, we will apply the current shared masks in forward and backward passes to train learnable shared masks. However, the masking operation is not differentiable since it zeros out all corresponding attention heads and set them to zero as inactive heads during forward passes. To solve this, we applied STE, which stands for straight-through estimator techniques to estimate gradients during backward passes so that the scores can be updated properly. Therefore, the most important neuron or attention heads must have the highest score theoretically due to natural selection. Additionally, we implemented a dynamic threshold scheduler for the movement pruning trainer. For instance, in our setup, we aim to prune 30% of the attention heads in the Transformer and 60% of the neurons in FFN layers. To minimize abrupt performance degradation, the scheduler gradually increases the total masking threshold from 0% to 30% following a cubic curve for masked attention heads. The thresholds will reach their peek at 85% of the training progress and stay at the peek until the end and use the remaining 15% of the progress for fine-tuning, as shown in Figure 3.

$$\text{threshold} = \text{threshold}_{\text{final}} - (\text{threshold}_{\text{final}} - \text{threshold}_{\text{init}}) \cdot (1 - progress)^3 \tag{3}$$

Figure 4 shows a sample pruning result for an FFN layer on the left, alongside the results for pruning self-attention and cross-attention in a decoder layer. Darker colors indicate that the corresponding neuron or attention head has been pruned.

5.3 Evaluations

We applied only our pruner on the fine-tuned BART-large-CNN model without applying distillation to show pruning results. The table 3 below shows 35% of storage reduction and about 4% of rouge score loss, which means our pruning strategy works. A reflection on the pruning strategy is that, we use learnable scores (masks), train those masks, and apply those masks during training to get the most valuable neurons / attention headers, which has similar semantics as QAT using quantization-aware weights during training. Besides, the STE strategy is applied as well both in pruning and in QAT.

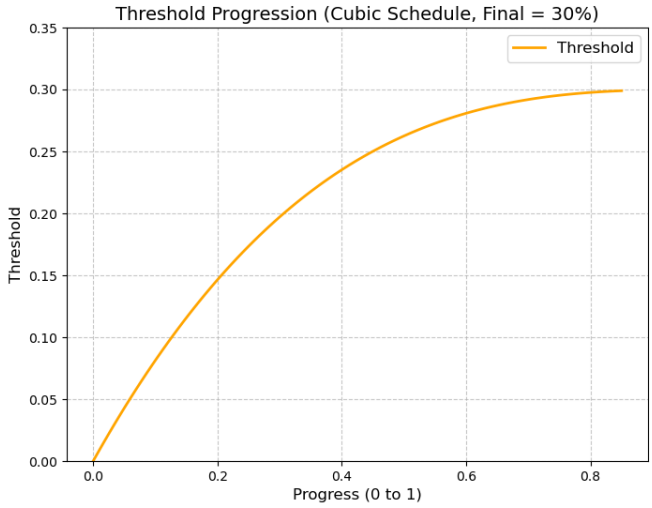
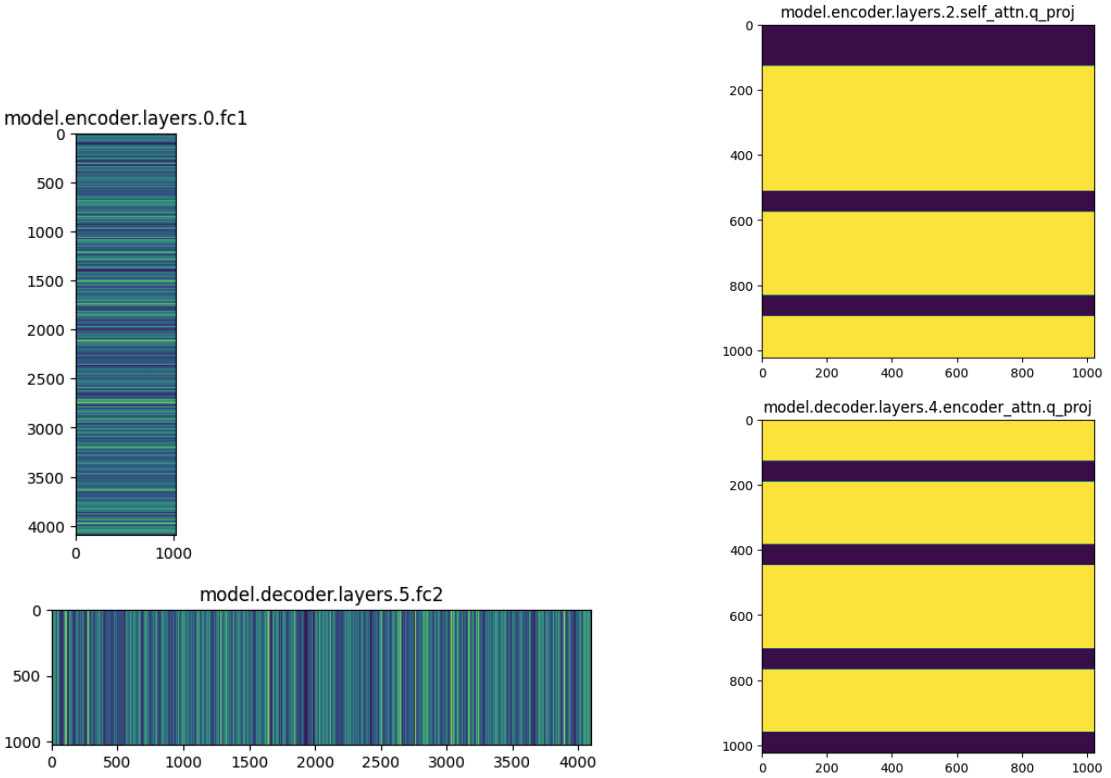


Figure 3: Movement Threshold Scheduler for Attention Layers



(a) Sample Neuron Pruning

(b) Sample Attention Head Pruning

Figure 4: Samples: Movement Pruning Distribution

The next table 4 shows the pruning result of the GPT-2 model, with a 25% reduction in storage and an increase in perplexity from 18.25 to 39.84. The reduction of model size reduction is smaller for GPT-2 compared to BART-large-CNN, as decoder layers in BART-large-CNN include both self-attention and cross-

Name	**BART-large-CNN**	
Metric	**Original**	**Pruned**
Encoder Layers	12	12
Decoder Layers	12	12
ROUGE-1 Score	0.717	0.713
ROUGE-2 Score	0.617	0.612
ROUGE-L Score	0.541	0.530
ROUGE-Lsum Score	0.559	0.535
Inference Time	59.25s	59.2s
Model Size	1.51GB	0.98GB

Table 3: Comparison of Original and Pruned Models for BART-large-CNN

attention, which can be pruned, whereas only self-attention is prunable in GPT-2's decoder layers. Hence the storage reduction on BART-large-CNN is greater.

Name	**GPT-2**	
Metric	**Original**	**Pruned**
Decoder Layers	12	12
Perplexity	18.25	39.84
Inference Time	43.18s	42.1s
Model Size	0.548GB	0.406GB

Table 4: Comparison of Original and Pruned Models for GPT-2

6 Quantization

6.1 Introduction

Quantization can reduce the precision of numbers used to present parameters or activations. By converting high-precision floating-point numbers (FP32, FP64) to lower precision (INT8, FP16), quantization reduces the memory footprint and computational requirements of a model, making it more efficient for deployment on resource-constrained devices like edge or mobile devices. Types of quantization include PTQ, standing for post-training quantization, and QAT, short for quantization-aware training. To map FP32, for instance, to INT8, we can convert floating point values into quantized space using linear mapping strategy, as expressed below:

$$r = S(q - Z) \tag{4}$$

$$q = \text{round}\left(\frac{r}{S} + Z\right) \tag{5}$$

, where r and q are the number before and after quantization; S and Z are scale and zero-point. The linear mapping is referred to as symmetric or asymmetric mapping, depending on whether Z is zero, as shown in Figure 5, where a symmetric range is converted to a symmetric INT8 range on the left and an asymmetric range is mapped to an asymmetric INT8 range.

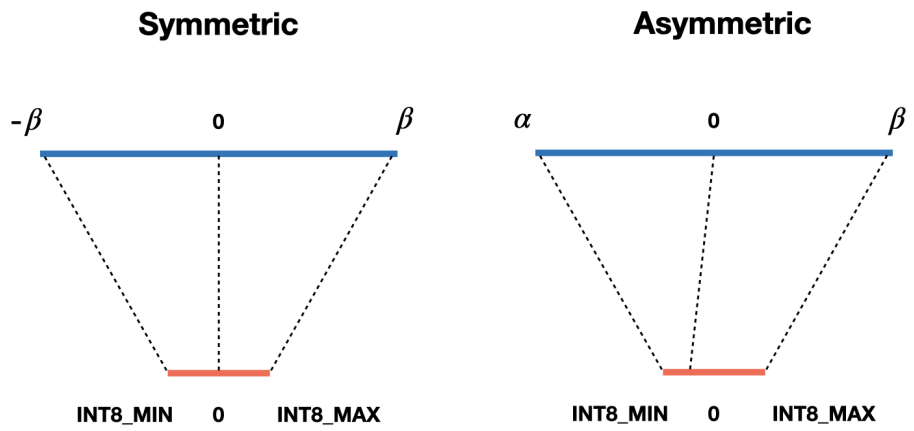


Figure 5: Linear Mapping with or without Zero-point

After mapping FP32 to INT8, which is PTQ, the memory footprint is significantly reduced; however, this comes at the cost of precision and performance degradation. To compensate for the degradation, calibration can be performed by feeding real data into the quantized model to estimate and adjust weight ranges.

Another approach for the model to adjust for quantization errors is to use online quantization techniques, for instance, QAT, instead of the offline PTQ method. QAT involves training the model with quantization simulated during the process to improve performance after quantization. Despite weights being quantized to INT8, during the forward process weights are still represented as FP32 to adjust for quantization errors. After quantization-aware training, the model learns to compensate for quantization errors, ensuring minimal performance degradation, followed by PTQ after training to convert the model to an INT8 format, and inference is performed using INT8 precision.

6.2 Implementation

Our approach follows the above description, performing QAT first, followed by PTQ. To implement QAT, We designed a `QLinear` layer to substitute the original `torch.nn.Linear` layers to perform quantization-aware training. There was a similar problem as described in the pruning section: the linear mapping process, as we call it `affine`, leveraging `round` and `clamp` operations, is non-differentiable. We applied the same approach described above: STE to estimate the backward gradient for our `QLinear` layers to learn and gradually adjust for quantization errors. We rewrote the backward function for the $XW + b$ linear operation, plugging in `affine` and `deaffine` logic for W parameters during forward and backward passes. Another operation worth mentioning is that we did not assign zero-points in `affine` and `deaffine` since during experiments we found the performance of the models was better using symmetric quantization perhaps due to weight distribution in the pre-trained models.

6.3 Evaluations

We applied only our quantizer on models, and the table 3 below shows 62% of storage reduction with a loss of less than 2% on the BART-large-CNN model. However, the inference time remains the same: after PTQ weights are transformed into quantized INT8 formats, and the model should have applied INT8 by INT8 matrix multiplication when doing linear calculation; however, it is challenging to find highly optimized INT8 by INT8 matrix third-party APIs both working on CUDA chips and MPS devices (Macbook M2). To keep the performance data consistent on both platforms, we employed a straightforward approach by converting INT8 to FP32 during inference. While the results remain consistent, this approach is slower. The results of GPT-2 are comparable to those of BART-large-CNN; therefore, they are not presented here for brevity.

Name	**BART-large-CNN**	
Metric	**Original**	**Quantized**
Encoder Layers	12	12
Decoder Layers	12	12
ROUGE-1 Score	0.717	0.717
ROUGE-2 Score	0.617	0.616
ROUGE-L Score	0.541	0.536
ROUGE-Lsum Score	0.559	0.552
Inference Time	59.25s	59.3s
Model Size	1.51GB	0.58GB

Table 5: Comparison of Original and Quantized Models for BART-large-CNN

7 Pipelines

7.1 Implementation

We combined these optimizations in a distillation-pruning-quantization order as a pipeline and performed the final optimized models on text summarization and text prediction. After each optimization was done, we performed model size calculation and performance measurement for that optimization to observe the effect.

7.2 Evaluation

The table 6 presents the results of the BART-large-CNN pipeline, which reduces the memory footprint by 77.5% and inference time by 26.2%, with a performance loss of less than 3.5%. However, there is still more space to optimize storage and inference time. For example, we can extract 3 or 4 layers from the teacher during distillation instead of 6, drop attention heads by more than 30% as we set, or implement INT8 by INT8 matrix multiplication logic for quantized model inference, but we would also like to balance the loss to get a good performance figure so we made some trade-offs on hyperparameter selections. From the table, we may find that the attention layers and the FFN layers have been sufficiently optimized, and the remaining storage is mainly due to the existence of the shared embedding layer and the language modeling head layers, which are both huge with a size of (50264×1024) in BART-large-CNN, indicating that these two layers, i.e. the vocabulary size is the current bottleneck. In comparison, unoptimized Q projection in BART-large-CNN only has a size of (1024×1024). One way to handle this is to prune the vocabulary table: we can scan all data from datasets and drop never-used words from the vocabulary table. We researched and found using

our BBC dataset, we could shrink the size of the vocabulary table to nearly half of the original. However, since this is highly dataset-related, so we finally did not apply it.

Pipeline	Fine-Tuned	Distilled	Pruned	Quantized	Percentage (Q / FT)
Encoder Layers	12	6	6	6	-
Decoder Layers	12	6	6	6	-
ROUGE-1 Score	0.717	0.716	0.712	0.707	98.6%
ROUGE-2 Score	0.617	0.617	0.611	0.604	97.8%
ROUGE-L Score	0.541	0.551	0.528	0.523	96.6%
ROUGE-Lsum Score	0.599	0.604	0.578	0.579	96.6%
Inference Time (s)	59.25	43.42	43.44	43.75	73.8%
Model Size (GB)	1.51	0.86	0.56	0.34	22.5%

Table 6: Pipeline of BART-large-CNN

For the GPT-2 pipeline, the results are as the below table 7, showing that the pipeline reduces the memory footprint by 69% and inference time by 21.5%, with an increase of perplexity of 47.41%. Therefore, the results in BART-large-CNN and GPT-2 optimizations are consistent and effective.

Pipeline	Fine-Tuned	Distilled	Pruned	Quantized	Percentage (Q / FT)
Decoder Layers	12	6	6	6	-
Perplexity	18.25	31.98	68.27	65.66	+47.41
Inference Time (s)	43.18	34.95	34.96	33.92	78.5%
Model Size (GB)	0.548	0.31	0.23	0.17	31.0%

Table 7: Pipeline of GPT-2

8 Conclusions and Future Work

Given the results shown above, for our proposed optimizations and pipelines, we believe the results still have potential for further enhancement, beyond the above-mentioned improvements. For instance, ONNX provides a framework for efficiently accelerating INT8 matrix multiplication, offering strong compatibility with various runtimes. Perhaps ONNX can help achieve faster inference speeds using quantized models. Additionally, our machine learning process is based entirely on the Python programming language, which is relatively slow in performance since Python only has an interpreter. Although most critical operations in PyTorch are written in pure C++, we still cannot ignore the penalty caused by Python itself. To compensate for this penalty, perhaps we can leverage JIT compilations embedded in PyTorch and compile the models into TorchScripts, thus avoiding the Python interpreter as much as possible to accelerate. Last but not least, regarding the fine-tuning process, perhaps we can apply LORA (low-rank adaptation), which splits weights into two low-rank matrices to accelerate gradient updates during backward propagation across layers without losing much precision. In summary, future optimizations should focus on exploring more sophisticated methods that balance computational efficiency and model precision and leveraging emerging advancements in model compression to push the boundaries of the performance of lightweight Transformers.

References

[1] K. Gokaslan and Others, "OpenWebText: An open-source replication of OpenAI's WebText dataset," available at `https://github.com/skylion007/openwebtext`, 2019.

[2] Wolf, Thomas, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, et al. "Transformers: State-of-the-Art Natural Language Processing." *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pp. 38–45. Online: Association for Computational Linguistics, 2020. Available: `https://aclanthology.org/2020.emnlp-demos.6`.

[3] G. Hinton, O. Vinyals, and J. Dean, "Distilling the Knowledge in a Neural Network," *arXiv preprint arXiv:1503.02531*, 2015. `https://arxiv.org/abs/1503.02531`

[4] Xinlei Chen, Simon Kornblith, Mohammad Noroozi, Geoffrey Hinton, *A Simple Framework for Contrastive Learning of Visual Representations*, arXiv preprint arXiv:2010.13002, 2020.

[5] M. Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Veselin Stoyanov, and Luke Zettlemoyer, *BART: Denoising Sequence-to-Sequence Pre-training for Natural Language Generation, Translation, and Comprehension*, arXiv preprint arXiv:1910.13461, 2020.

[6] Alec Radford, Jeff Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever, *Language Models are Unsupervised Multitask Learners*, 2019.