Lightweight Transformer-Based Context-Aware Text Generation

Guangyan An*
Department of CISE
University of Florida
Gainesville, USA
anguangyan@ufl.edu

Xiaolin Zheng*
Department of CISE
University of Florida
Gainesville, USA
xiaolinzheng@ufl.edu

I. 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 quantization. In the following sections, we will present the development, internals, and outcomes of the optimized pre-trained models. Section II discusses the related work. Section III presents knowledge distillation, movement pruning, and quantization (including post-training quantization, PTQ and quantization-aware training, QAT) methodology. Section IV shares the knowledge distiller, movement pruning techniques, and quantization implementations. It also 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 V concludes this project and discusses various potential future improvement we have observed and summarized in these months during the project.

* These authors contributed equally to this research.

II. Related Work

A. Knowledge Distillation

Hinton et al. proposed the concept of knowledge distillation [1], providing the basis for model compression and optimization. Knowledge distillation uses soft targets of the teacher model as the learning target for the smaller student model by minimizing the KL divergence between the output probability distributions of the teacher and the student. After that, FitNets [2] proposed using intermediate representations, such as feature maps and hidden states, of the teacher model as a guidance for students to learn internal representations. Students not only imitate the final outputs of the teachers, but also replicate their details. Then, DistilBERT [3] applied this technique to BERT, a Transformer model, reducing its parameters by 40% percent while retaining more than 97% of the performance on various workloads. Subsequently, Shleifer and Rush [4] extended knowledge distillation techniques to the field of text summarizations. Their approach introduced task-related techniques to distill large summarization models, such as T5 and BART, into smaller models, with a smaller model inference time. These researches demonstrate that by effectively extracting and transferring knowledge from different layers of the teacher model, knowledge distillation can significantly reduce model footprint and inference time, making models adaptable to various application scenarios.

B. Pruning

Early pruning strategy focused on sparsifying weight matrices by removing weights with small values, which is unstructured pruning [5]. For Transformers, structured pruning is applied to optimize the self-attention and feed-forward neural network components. This approach calculates importance scores to select necessary attention heads and drop the others [6] without significantly degrading the model performance. After that, movement pruning [7] was proposed to dynamically adjust the pruning strategy during training or fine-tuning process according to gradient information, improving the overall model precision and robustness after pruning. This strategy can be applied to both unstructured and structured pruning, which is flexible. Famous real-world model pruning implementations include `torch.nn.utils.prune`, which contains both unstructured and structured pruning. For unstructured pruning, the implementation leverages random masks or weight thresholds to drop weights in a parameter-by-parameter manner. Structured pruning is a larger-granularity pruning that removes neurons, convolution kernels, or attention heads.

C. Quantization

Quantization includes weight quantization and activation quantization. In recent years, 8-bit weight and activation quantization has become mainstream in Transformer models, since quantizing a 32-bit float point Transformer to an 8-bit model can largely reduce the model footprint and inference cost while retaining high precision. For instance, Q8BERT [8] introduced a quantized 8-bit BERT model, reducing the model size by approximately 4x by applying quantization-aware training. Furthermore, TernaryBERT [9] uses a ternary quantization mechanism, and the weights of the model only take values in $\{-1, 0, 1\}$, further achieving a higher compression rate while retaining the performance. Real-world implementations of quantization include TorchQuantization, which provides implementations of post-training quantization and quantization-aware training. Post-training quantization requires the offline collection of calibration data to generate precise scale factors after converting the model. For quantization-aware training, TorchQuantization supports inserting fake quantization modules in the model and significantly improves the performance of low-precision models by simulating and propagating quantized model loss to optimize the weights of the quantized model.

III. Method

A. Experiment Setup

We used a subset of OpenWebText dataset [10], 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* [11] 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. 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.

B. Knowledge Distillation

Knowledge Distillation 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:

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

where $L_{\text{CE}}(y, \hat{y})$ 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.

BART-large-CNN [12] 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 [13] 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 [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

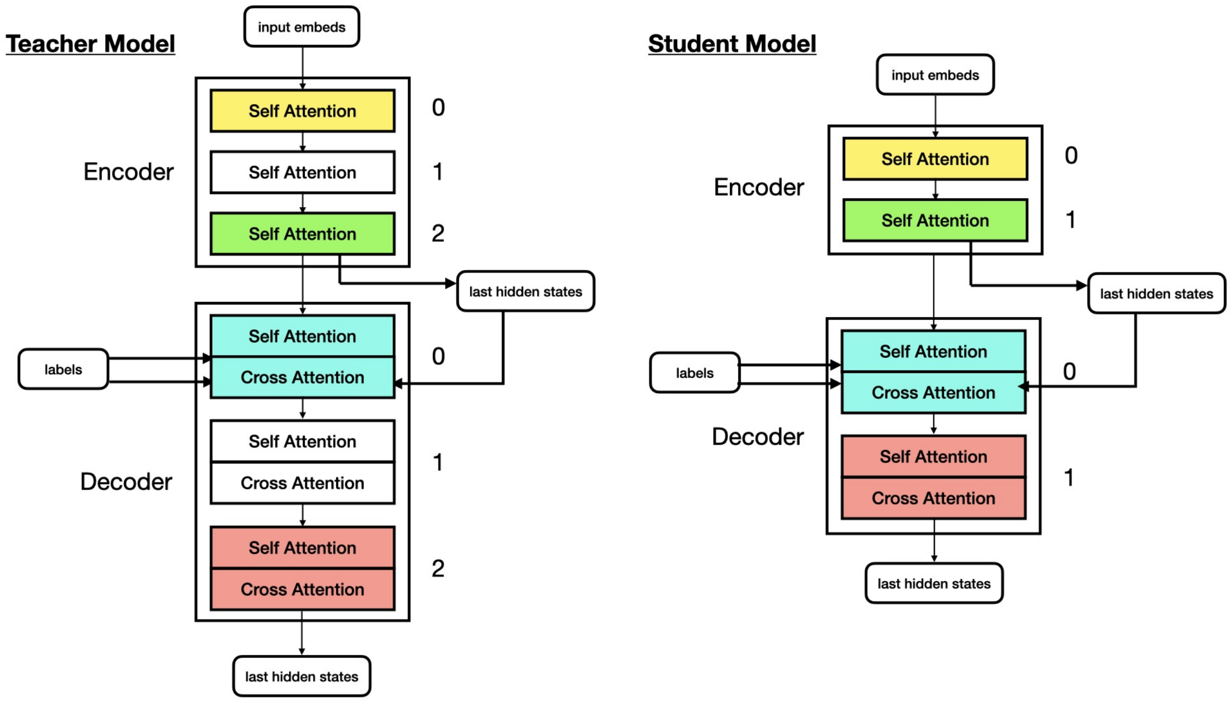

Fig. 1: Example of distillation of Encoder-Decoder model, e.g. BART-large

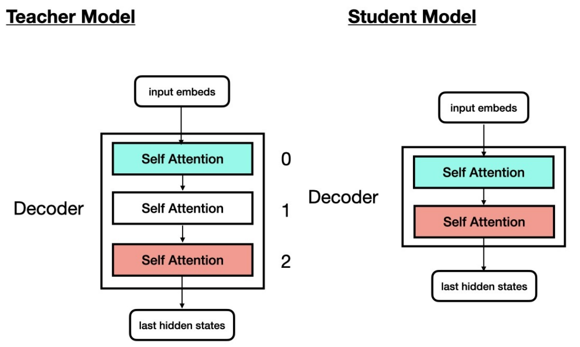

Fig. 2: Example of distillation of Decoder-Only model, e.g. GPT-2

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.

C. Movement Pruning

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.

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 threshold T for the movement pruning trainer, as shown in Eq. 3. 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.

$$T = T_{\text{init}} + (T_{\text{final}} - T_{\text{init}}) \times (1 - \text{progress})^3 \tag{3}$$

Figures 4 and 5 show pruning results for an FFN layer and a decoder layer, respectively. In Figure 4, darker colors indicate the neurons that have been pruned, while in Figure 5, darker colors highlight pruned self-attention and cross-attention heads.

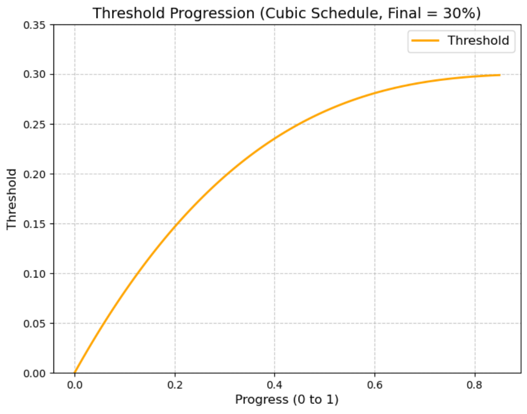

Fig. 3: Movement Threshold Scheduler for Attention Layers

D. Quantization

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:

$$\begin{aligned} r &= S(q - Z) \\ q &= \text{round}\left(\frac{r}{S} + Z\right) \end{aligned} \tag{4}$$

where r and q are the number before and after quantization; S and Z are scale and zero-point. The linear mapping is

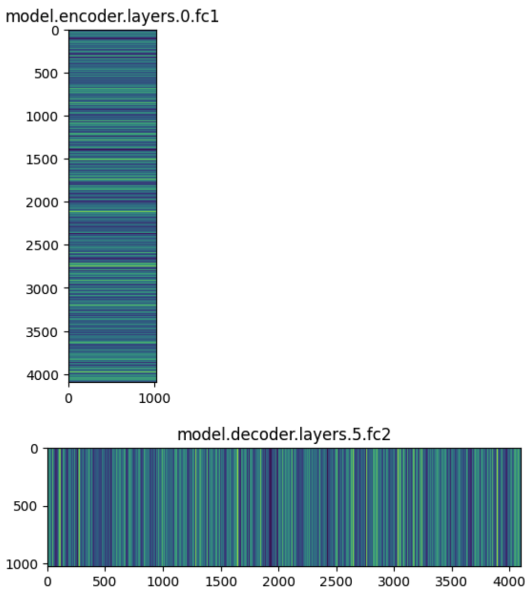

Fig. 4: Sample Neuron Pruning for FFN

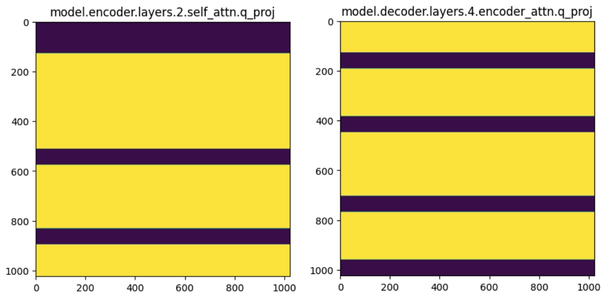

Fig. 5: Sample Attention Head Pruning for Self-Attention and Cross-Attention in Decoder.

referred to as symmetric or asymmetric mapping, depending on whether Z is zero, as shown in Figure 6, 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.

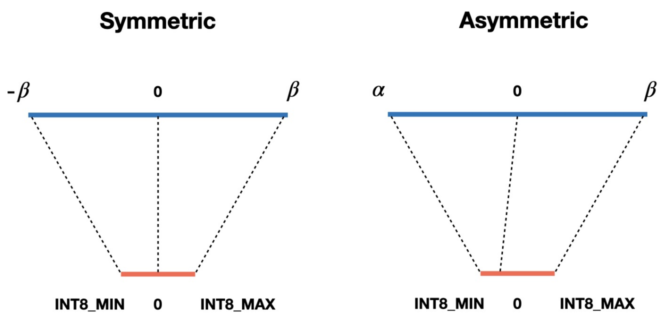

Fig. 6: Linear Mapping with or without Zero-point

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.

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.

IV. Experiment Results

A. Knowledge Distillation

We have evaluated the performance, inference time, and the reduction of storage for our distiller with the settings discussed above. As the table I 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 I: Comparison of Teacher and Student Models for BART-large-CNN

The result of distiller for GPT-2 are shown below in the table II. 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 II: Comparison of Teacher and Student Models for GPT-2

B. Movement Pruning

We applied only our pruner on the fine-tuned BART-large-CNN model without applying distillation to show pruning results. The table III 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.

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

The next table IV 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-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 IV: Comparison of Original and Pruned Models for GPT-2

C. Quantization

We applied only our quantizer on models, and the table III 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 V: Comparison of Original and Quantized Models for BART-large-CNN

D. Pipelines

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.

The table VI 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.

For the GPT-2 pipeline, the results are as the below table VII, 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.

V. 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.

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 VI: Pipeline of BART-large-CNN

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 VII: Pipeline of GPT-2

References

[1] Geoffrey Hinton. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015.

[2] Adriana Romero, Nicolas Ballas, Samira Ebrahimi Kahou, Antoine Chassang, Carlo Gatta, and Yoshua Bengio. Fitnets: Hints for thin deep nets. *arXiv preprint arXiv:1412.6550*, 2014.

[3] V Sanh. Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter. *arXiv preprint arXiv:1910.01108*, 2019.

[4] Sam Shleifer and Alexander M Rush. Pre-trained summarization distillation. *arXiv preprint arXiv:2010.13002*, 2020.

[5] Song Han, Jeff Pool, John Tran, and William Dally. Learning both weights and connections for efficient neural network. *Advances in neural information processing systems*, 28, 2015.

[6] Paul Michel, Omer Levy, and Graham Neubig. Are sixteen heads really better than one? *Advances in neural information processing systems*, 32, 2019.

[7] Victor Sanh, Thomas Wolf, and Alexander Rush. Movement pruning: Adaptive sparsity by fine-tuning. *Advances in neural information processing systems*, 33:20378–20389, 2020.

[8] Ofir Zafrir, Guy Boudoukh, Peter Izsak, and Moshe Wasserblat. Q8bert: Quantized 8bit bert. In *2019 Fifth Workshop on Energy Efficient Machine Learning and Cognitive Computing-NeurIPS Edition (EMC2-NIPS)*, pages 36–39. IEEE, 2019.

[9] Wei Zhang, Lu Hou, Yichun Yin, Lifeng Shang, Xiao Chen, Xin Jiang, and Qun Liu. Ternarybert: Distillation-aware ultra-low bit bert. *arXiv preprint arXiv:2009.12812*, 2020.

[10] Aaron Gokaslan and Vanya Cohen. Openwebtext corpus, 2019.

[11] Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, et al. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 conference on empirical methods in natural language processing: system demonstrations*, pages 38–45, 2020.

[12] Mike 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*, 2019.

[13] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.

Appendix A
Link to Code Repository

The GitHub repository containing the code is available at Phoslight/CAP6617-Project