OPT2025: 17th Annual Workshop on Optimization for Machine Learning

Projected Compression: Trainable Projection for Efficient Transformer Compression

Maciej Stefaniak M.STEFANIAK8@UW.EDU.PL
Michał Krutul
Mikołaj Dziok
Jan Małaśnicki
Maciej Pióro
Jakub Krajewski
Sebastian Jaszczur
Marek Cygan
Kamil Adamczewski
Jan Ludziejewski
University of Warsaw, Ideas NCBR, Wrocław University of Science and Technology, Poland

Abstract

Large language models have steadily increased in size to achieve improved performance; however, this growth has also led to greater inference time and computational demands. Consequently, there is rising interest in model size reduction methods. To address this issue, we propose **Projected Compression**, a novel model compression technique, that reduces model weights by utilizing projection modules. Specifically, we first train additional trainable projections weights and preserve access to all the original model parameters. Subsequently, these projections are merged into a lower-dimensional product matrix, resulting in a reduced-size standard Transformer-based model. Unlike alternative approaches that require additional computational overhead, our method matches the base model's per-token computation step in FLOPs. Experimental results show that Projected Compression outperforms the comparable pruning compression method on higher quality models. Moreover, the performance margin scales well with the number of tokens.

1. Introduction

As large language models (LLMs), exceptionally good at Natural Language Processing (NLP) tasks [2, 15, 18], continue to be developed at increasing scale, their computational and memory requirements present growing challenges for deployment, experimentation, and fine-tuning [4, 19]. Reducing the size and operational costs of the model while preserving quality is, therefore, essential to make LLMs more widely accessible to the research community and the public. Model compression techniques have emerged as a popular solution to this problem, with pruning remaining one of the most widely adopted approaches due to its simplicity and effectiveness. However, standard hard pruning methods suffer from an inherent limitation: once parameters are removed, their representational capacity is permanently lost, often leading to performance degradation. As a result, hard pruned models usually require additional retraining.

Motivated by these advances, we propose Projected Compression (PC), a novel compression method that preserves access to all original model parameters through gradient-optimized projection modules. Rather than removing unimportant weights, PC redirects their influence through learnable

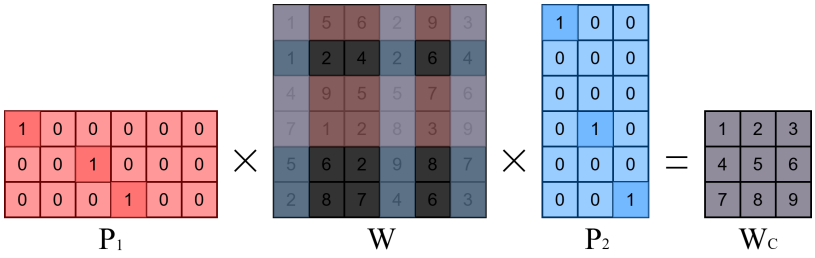

Figure 1: Simplified illustration of a projection module, where P_1 and P_2 are projection matrices, W is a frozen base model parameters and W_C is compressed weights matrix.

projection modules. Thus, each weight matrix is accompanied with two matrices that produce a lower-rank matrix which is used to build a new compressed architecture.

Furthermore, the projection modules can be initialized using custom schemes, most notably by weight importance, which allows the model to gradually reincorporate useful dimensions during training, while still achieving the same cost per processed token as a vanilla transformer. This is because updating the additional projection weights and projecting the base model weights into a lower dimension incur costs that are independent of batch size, and with sufficiently large batches, these costs become negligible.

We apply PC to Transformer-based language models. The preliminary results show that the PC improves performance against hard pruning and retraining given the same computational budget.

To the best of our knowledge, Projected Compression is the first method to reframe model compression as a trainable projection problem operating on a frozen base model that takes advantage of additional weights importance information, explicitly targeting transformer core matrices under structured sparsity constraints. The main contributions of this work are:

- Introduction of PC, a novel compression method that retains access to frozen base weights via trainable projection modules, while maintaining the same training cost per token as standard transformer.
- Experimental results demonstrating performance gains of Projected Compression over hard pruning increase with: (1) quality of the base model relative to number of parameters, (2) base model size, and (3) training cost to obtain a compressed model.

2. Related Work

Pruning. A widely adopted technique for compressing deep neural networks, model pruning removes weights considered less important for model performance [8, 10, 13]. Hard pruning methods are based on importance criteria, most commonly magnitude-based thresholds [8] or activation statistics [12]. Although efficient and simple, upon pruning these methods permanently discard information . Projected Compression uses for initialization the same weight-importance criteria as hard pruning for parameters removal.

On the other hand, in soft pruning [6, 16, 20], the masked weights remain in memory and continue to receive gradients, allowing their reemergence if they regain significance. Projected Compression similarly enables all weights to influence model compression process mixing their influence by gradient optimized projections.

Parameter-efficient fine-tuning. Parameter-efficient fine-tuning (PEFT) approaches, such as LoRA, Prefix Tuning, BitFit, and QLoRA [1, 5, 11, 14], enable model adaptation by freezing the base model weights and introducing a small set of trainable parameters. These methods are designed to efficiently adapt large models to downstream tasks under strict memory or computational constraints, often using lightweight modules such as low-rank adapters. Projected Compression is conceptually related to LoRA and other PEFT approaches by freezing base weights and introducing trainable modules, projection matrices, which adapt the behavior of the projected model without modifying its base model core parameters.

Other work. In a concurrent work, Hao et al. [9] introduced a low-rank clone (LRC) method for compression using low-rank weights that are learned with the use of distillation loss during retraining. Contrary, PC uses weights importance projections initialization similar to dimension reduction operation. Our method is additionally much cheaper matching cost per processed token in retraining with vanilla transformer.

3. Method

Projected Compression (PC) is a structured model compression method that uses *projection modules* to construct a lower-dimensional representation of a Transformer model linear layers while retaining access to its full set of frozen base parameters. PC introduces additional learnable projection matrices that operate over the original weights. The compressed model is fully defined by these projections, which are gradient optimized during training, while the original base model weights remain frozen.

3.1. Projection-Based Structured Compression

Projected Compression targets structured dimension reductions in the Transformer architecture, specifically the *feedforward hidden size* and *embedding dimension* (model width). For each frozen base weight matrix $W \in \mathbb{R}^{d^{\text{in}} \times d^{\text{out}}}$, we are projecting d^{in} and d^{out} dimensions to a significantly smaller d_S^{in} and d_S^{out}. Corresponding projection module is presented in Figure 1. This module consists of one or two trainable projection matrices, P_1 and/or P_2, which produce a compressed weight $W_C \in \mathbb{R}^{d_S^{\text{in}} \times d_S^{\text{out}}}$ of reduced rank:

$$W_C = P_1 W P_2$$

where:

- $W \in \mathbb{R}^{d^{\text{in}} \times d^{\text{out}}}$ — the frozen base model weights,
- $P_1 \in \mathbb{R}^{d_S^{\text{in}} \times d^{\text{in}}}$ — a projection that downsamples the input dimension,
- $P_2 \in \mathbb{R}^{d^{\text{out}} \times d_S^{\text{out}}}$ — a projection that downsamples the output dimension,
- $W_C \in \mathbb{R}^{d_S^{\text{in}} \times d_S^{\text{out}}}$ — the resulting compressed weights.

During the course of training, only the projection matrices P_1 and P_2 are updated by gradient descent, while the base weights W remains frozen. This allows frozen weights to be blended into the active computation subspace, offering the potential to recover and leverage information from parameters that would have been permanently removed in standard pruning.

This mechanism preserves compatibility with the standard Transformer architecture. The projected weights W_C are recomputed before each forward pass and are treated as regular layer parameters, ensuring full compatibility with the existing model infrastructure and methods such as an addition of distillation logits loss.

Importantly, this construction is functionally equivalent to a two-step transformation of the input token representation. For a given input token vector x, multiplying it by compressed weights W_C (that is, computing xW_C) is equivalent to first projecting x up into the base model space via P_1, performing a matrix multiplication with the full frozen weight matrix W, and then projecting the result back down via P_2:

$$xW_C = ((xP_1)W)P_2$$

This interpretation highlights that Projected Compression allows each token representation to interact with the full capacity of the base model during training, even though only a compressed projection is used during inference. It reinforces the intuition that PC does not discard any information a priori but instead passes token activations through learnable subspaces of the frozen parameter space.

One-sided Projection. In general, Projection Compression has two projection matrices but one may also consider to use only one projection matrix. In the one-sided variant, only P_1 or P_2 is used, depending on the compression axis.

Residual Weights Extension. To improve late-stage flexibility during training, a residual term $W_r \in \mathbb{R}^{d_S^{\text{out}} \times d_S^{\text{in}}}$ is added to the compressed weights:

$$W_C = P_1 W P_2 + W_r$$

The residual weights W_r are initialized with zeros and trained alongside P_1 and P_2. This addition helps the model maintain the optimization flexibility as it diverges from the frozen base.

3.2. Initialization

The initialization of projection matrices plays important role in the quality of the compressed model. In this work, we initialize the matrices with importance scores derived from standard pruning heuristics (e.g., magnitude-based criteria). This ensures that the most important weights dominate the initial projection, while the less important dimensions are initially suppressed. Unlike pruning, however, Projected Compression does not eliminate any parameters - less important dimensions are excluded from the initial projection but remain accessible. As training progresses, these suppressed components gradually contribute to the model through optimization of the projection weights.

3.3. Training Dynamics and Resource Efficiency

One of the key advantages of Projected Compression is its training computing efficiency. Forward/backward over tokens runs through the compressed matrix, W_C, not through the full W. Gradients wrt activations accumulate into the small parameter space of W_C. After the step, those are mapped to gradients for P_1, P_2 via W (which stays frozen). This preserves access to the full model during training but keeps the per-token computations small.

There are two for costs each optimizer step:

1. Fixed (token-independent) cost rebuilding $W_C = P_1 W P_2$ and backpropagating from ∇_{W_C} to $\nabla_{P_1}, \nabla_{P_2}$ touch the full width d. This cost does not scale with batch/sequence length (tokens's processed)—it's a one-off per step (or per rebuild).

2. Token-dependent cost of forward and backward through W_C scales with the number of tokens processed in the step (batch). This part is the same order as retraining a hard-pruned model of the compressed width.

As a result, as you increase tokens per step, the fixed projection cost is amortized and becomes negligible relative to the token-dependent work. Consequently, the overall cost per optimization step is equivalent to that of retraining a compressed transformer model obtained through hard pruning.

Limitations. We should note that the proposed method comes at the expense of additional memory usage: the optimizer must store both the trainable projection matrices and the full set of frozen base model weights. This overhead can be mitigated through memory saving techniques such as gradient checkpointing [3] and offloading inactive projection modules to CPU memory during training.

4. Experiments

4.1. Experimental setup

Pre-trained models. As an initial step, we trained a series of GPT-2 [17] style models to serve as base models for subsequent pruning experiments. Specifically, we used two model configurations: a 300M parameter model (16 layers, 16 attention heads) and an 800M parameter model (24 layers, 24 attention heads). For each configuration, models were trained on varying amounts of data to achieve token-to-parameter ratios of 20:1 and 80:1, respectively.

Evaluation. To assess the effectiveness of the proposed PC method, we conducted a comparative evaluation against the pruning baseline. For each of the pre-trained base models described above, covering two model sizes and two token-to-parameter ratios, we ran a series of experiments across varying numbers of training tokens and compression levels. Both compression techniques were applied to each setting and performance was measured using the Cross-Entropy loss averaged over the last 100 training steps.

Training setting. Experiments were conducted across multiple clusters, with compute nodes equipped with 4× NVIDIA GH200, H100, or A100 GPUs. All models were trained on the same sequences sampled from C4 dataset [18].

4.2. Results

The experiments investigate scaling trends by comparing two compression pipelines: Hard Pruning with Retraining (HPR) and Projected Compression (PC). Both methods use identical weight-importance scores to guide pruning decisions and using the same number of training steps are matched in terms of compute.

Figure 2 presents the main results comparing our proposed PC method with the Hard Pruning baseline at 50% compression. Across all evaluated configurations, PC consistently yields lower Cross-Entropy loss than Hard Pruning on models trained with a higher token-to-parameter ratio and trained on more tokens. This trend highlights the strength of PC in preserving model quality under data-rich conditions. Additional results at other compression levels are provided in Table 1, which further confirm the robustness of PC across a range of compression settings. All conducted experiments can be seen in Appendix A.2.

4.3. Base model size and quality

The experimental trends suggest that Projected Compression is particularly well-suited for compressing larger and higher-quality base models. In particular, PC performs best when the model has been trained on a sufficiently large number of tokens relative to its parameter count. This is

Model N	Method	Tokens processed during training compression pipeline 1.25B	2.5B	5B	7.5B	10B
35% compression						
300M	HPR	**3.1759**	3.1362	3.1070	3.0925	3.0775
300M	PC	3.1776	**3.1361**	**3.1037**	**3.0882**	**3.0711**
800M	HPR	**2.9224**	2.8827	2.8519	2.8420	2.8259
800M	PC	3.0327	**2.8780**	**2.8465**	**2.8355**	**2.8192**
50% compression						
300M	HPR	**3.2688**	**3.2172**	3.1781	3.1577	3.1402
300M	PC	3.2823	3.2198	**3.1744**	**3.1528**	**3.1341**
800M	HPR	3.0286	2.9733	2.9300	2.9128	2.8930
800M	PC	**3,0217**	**2.9652**	**2.9203**	**2.9037**	**2.8840**
65% compression						
300M	HPR	**3.3753**	**3.3095**	3.2603	3.2392	3.2175
300M	PC	3.3858	3.3134	**3.2591**	**3.2341**	**3.2133**
800M	HPR	**3.1245**	**3.0569**	**3.0056**	2.9828	2.9622
800M	PC	3.1375	3.0626	3.0069	**2.9815**	**2.9591**

Table 1: Projected Compression (PC) vs Hard Pruning Retraining (HPR) - Cross-Entropy loss of base model compression of different sizes (Model N) pretrained with 80:1 tokens to parameters ratio. Best loss from each method pair is bolded.

illustrated in Figure 2, where Projected Compression yields clear improvements over hard pruning at a token-to-parameter ratio of 80:1. Similarly, the performance advantage of the PC is more pronounced in the 800M-parameter model than in the 300M-parameter variant, as shown in Table 1.

These observations support the view that Projected Compression is a more effective compression method for the types of models that are the most efficient and valuable to compress. This efficiency trend that compression is most beneficial when applied to large, high-quality base models in shown in Frantar et al. work [7].

5. Conclusion and Future Work

In this work, we presented Projected Compression, a novel compression technique that preserves access to all original model parameters through learnable projection weights. Experimental results demonstrate that our method is an effective method for compressing standard transformer-based models. Our results indicate that the effectiveness of Projected Compression relative to hard pruning and retraining increases with base model size, and its pretraining number of tokens. This work opens the possibility of adapting PC method to a broader range of architectures, which we leave for future exploration.

References

[1] Elad Ben Zaken, Shauli Ravfogel, and Yoav Goldberg. Bitfit: Simple parameter-efficient fine-tuning for transformer-based masked language-models, 2021.

[2] Rishi Bommasani, Drew A Hudson, Ehsan Adeli, Russ Altman, Simran Arora, Sydney von Arx, Michael S Bernstein, Jeannette Bohg, Antoine Bosselut, Emma Brunskill, et al. On the opportunities and risks of foundation models. *arXiv preprint arXiv:2108.07258*, 2021.

[3] Tianqi Chen, Bing Xu, Chiyuan Zhang, and Carlos Guestrin. Training deep nets with sublinear memory cost. *arXiv preprint arXiv:1604.06174*, 2016.

[4] Ben Cottier, Robi Rahman, Loredana Fattorini, Nestor Maslej, Tamay Besiroglu, and David Owen. The rising costs of training frontier ai models. *arXiv preprint arXiv:2405.21015*, 2024.

[5] Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. Qlora: Efficient finetuning of quantized llms. *arXiv preprint arXiv:2305.14314*, 2023.

[6] Utku Evci, Trevor Gale, Jacob Menick, Pablo Samuel Castro, and Erich Elsen. Rigging the lottery: Making all tickets winners. In *International Conference on Machine Learning (ICML)*, 2020. URL https://arxiv.org/abs/1911.11134.

[7] Elias Frantar, Carlos Riquelme, Neil Houlsby, Dan Alistarh, and Utku Evci. Scaling laws for sparsely-connected foundation models, 2023. URL https://arxiv.org/abs/2309.08520.

[8] 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.

[9] Jitai Hao, Qiang Huang, Hao Liu, Xinyan Xiao, Zhaochun Ren, and Jun Yu. A token is worth over 1,000 tokens: Efficient knowledge distillation through low-rank clone, 2025. URL https://arxiv.org/abs/2505.12781.

[10] Babak Hassibi and David Stork. Second order derivatives for network pruning: Optimal brain surgeon. *Advances in neural information processing systems*, 5, 1992.

[11] Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. Lora: Low-rank adaptation of large language models. *arXiv preprint arXiv:2106.09685*, 2021.

[12] Hengyuan Hu, Rui Peng, Yu-Wing Tai, and Chi-Keung Tang. Network trimming: A data-driven neuron pruning approach towards efficient deep architectures. *arXiv preprint arXiv:1607.03250*, 2016.

[13] Yann LeCun, John Denker, and Sara Solla. Optimal brain damage. *Advances in neural information processing systems*, 2, 1989.

[14] Xiang Lisa Li and Percy Liang. Prefix-tuning: Optimizing continuous prompts for generation. *arXiv preprint arXiv:2101.00190*, 2021.

[15] Ben Mann, N Ryder, M Subbiah, J Kaplan, P Dhariwal, A Neelakantan, P Shyam, G Sastry, A Askell, S Agarwal, et al. Language models are few-shot learners. *arXiv preprint arXiv:2005.14165*, 1:3, 2020.

[16] Decebal Constantin Mocanu, Elena Mocanu, Peter Stone, Phuong H Nguyen, Madeleine Gibescu, and Antonio Liotta. Sparse evolutionary deep learning with over one million artificial neurons on commodity hardware. *Neural Computing and Applications*, 32:2281–2301, 2020. doi: 10.1007/s00521-018-3735-8.

[17] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. *preprint*, 2018. URL https://cdn.openai.com/better-language-models/language_models_are_unsupervised_multitask_learners.pdf.

[18] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of machine learning research*, 21(140):1–67, 2020.

[19] Aditi Singh, Nirmal Prakashbhai Patel, Abul Ehtesham, Saket Kumar, and Tala Talaei Khoei. A survey of sustainability in large language models: Applications, economics, and challenges. In *2025 IEEE 15th Annual Computing and Communication Workshop and Conference (CCWC)*, pages 00008–00014. IEEE, 2025.

[20] Michael Zhu and Suyog Gupta. To prune, or not to prune: exploring the efficacy of pruning for model compression. *arXiv preprint arXiv:1710.01878*, 2017.

Appendix A. Experiments

A.1. Weights importances in experiments initialization

We have tested the influence on the importance of random and magnitude-based weights for tested base models in both tested compression methods. The magnitude-based method produced the worst performance than random pruning for each computation method Table 3.

A.2. All experiments

Model size (N)	Tokens ratio ($T : N$)	Loss
800M	80:1	2.7234
800M	20:1	2.8547
300M	80:1	3.0041
300M	20:1	3.1450

Table 2: Base models loss.

	Projected Compression		Hard Pruning Retraining	
Retraining tokens	2.5B	5B	2.5B	5B
random	**2.9652**	**2.9203**	**2.9733**	**2.9300**
magnitude	2.9655	2.9213	2.9764	2.9320

Table 3: Magnitude vs random based weights importance. Cross-Entropy loss when compressing 800M model by 50% (80:1 tokens to parameters ratio).

Model N	Ratio T:N	Method	Tokens processed during training compression pipeline 1.25B	2.5B	5B	7.5B	10B
			35% compression				
300M	20:1	HPR	**3.2645**	**3.2175**	**3.1816**	**3.1570**	**3.1339**
300M	20:1	PC	3.2683	3.2221	3.1845	3.1622	3.1400
300M	80:1	HPR	**3.1759**	3.1362	3.1070	3.0925	3.0775
300M	80:1	PC	3.1776	**3.1361**	**3.1037**	**3.0882**	**3.0711**
800M	20:1	HPR	**3.0074**	**2.9657**	**2.931**	**2.9179**	**2.9031**
800M	20:1	PC	3.0152	2.974	2.9380	2.9238	2.9056
800M	80:1	HPR	**2.9224**	2.8827	2.8519	2.8420	2.8259
800M	80:1	PC	3.0327	**2.8780**	**2.8465**	**2.8355**	**2.8192**
			50% compression				
300M	20:1	HPR	**3.3302**	**3.2741**	**3.2258**	**3.1980**	**3.1758**
300M	20:1	PC	3.3421	3.2742	3.2329	3.2070	3.1830
300M	80:1	HPR	**3.2688**	**3.2172**	3.1781	3.1577	3.1402
300M	80:1	PC	3.2823	3.2198	**3.1744**	**3.1528**	**3.1341**
800M	20:1	HPR	**3.0872**	**3.0345**	**2.9896**	**2.9695**	**2.9484**
800M	20:1	PC	3.0925	3.0380	2.9948	2.9752	2.9543
800M	80:1	HPR	3.0286	2.9733	2.9300	2.9128	2.8930
800M	80:1	PC	**3,0217**	**2.9652**	**2.9203**	**2.9037**	**2.8840**
			65% compression				
300M	20:1	HPR	**3.4125**	**3.3464**	**3.2919**	**3.2628**	**3.2455**
300M	20:1	PC	3.4241	3.356	3.3027	3.2755	3.2523
300M	80:1	HPR	**3.3753**	**3.3095**	3.2603	3.2392	3.2175
300M	80:1	PC	3.3858	3.3134	**3.2591**	**3.2341**	**3.2133**
800M	20:1	HPR	**3.1733**	**3.1098**	**3.0577**	**3.0341**	**3.0109**
800M	20:1	PC	3.1848	3.1188	3.0638	3.0380	3.0159
800M	80:1	HPR	**3.1245**	**3.0569**	**3.0056**	2.9828	2.9622
800M	80:1	PC	3.1375	3.0626	3.0069	**2.9815**	**2.9591**

Table 4: Projected Compression (PC) vs Hard Pruning Retraining (HPR) - Cross-Entropy loss of base model compression of different sizes (Model N) pretrained with different tokens to parameters ratio (Ratio T:N). Best loss from each method pair is bolded.

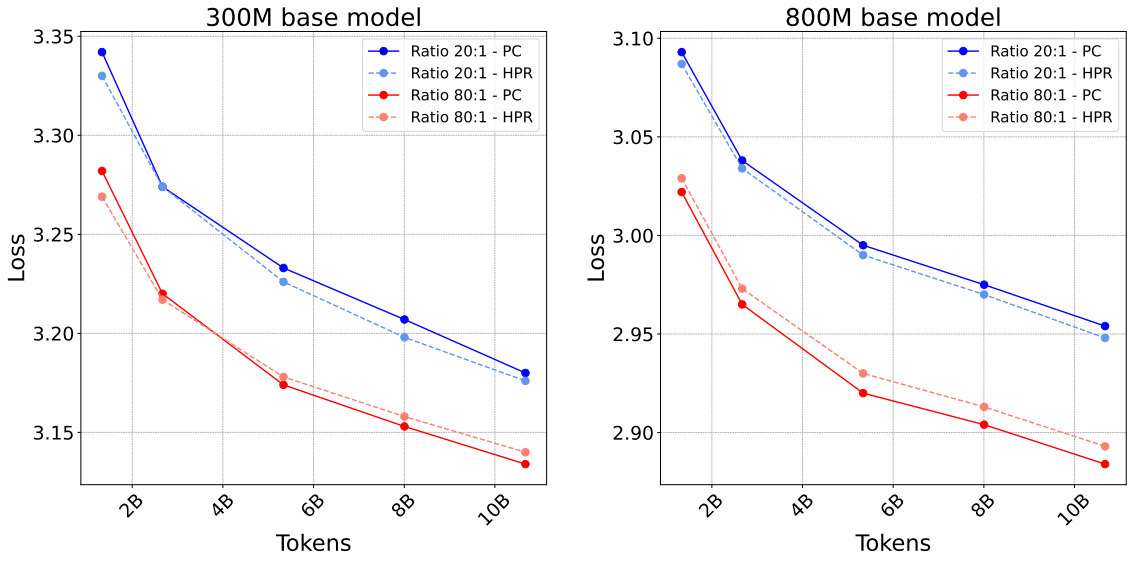

Figure 2: Compression of 50% parameters - Projected Compression (PC) vs Hard Pruning Retraining (HPR) for base models of different sizes pretrained with different tokens to parameters ratio (Ratio T:N).