EFFICIENT AND ROBUST WEB SCALE LANGUAGE MODEL BASED RETRIEVAL, GENERATION, AND UNDERSTANDING

BY

DANIEL CAMPOS

DISSERTATION

Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy in Computer Science
in the Graduate College of the
University of Illinois Urbana-Champaign, 2023

Urbana, Illinois

Doctoral Committee:

Professor Cheng Xiang Zhai
Dr. Alessandro Magnani (Walmart Labs, External Member)
Professor Jiawei Han
Professor Kevin Chang

ABSTRACT

Large language models effectively generate contextualized word representations across languages, domains, and tasks. Drive by these abilities, these models have become a building staple for many researchers and engineers who use text as their medium of representation, much like concrete is a staple in the construction world. Via the broad study and implementation, problems with large models have come to light: they can be expensive, brittle to noise, and produce unwanted outputs. Their large size and computational overhead make them difficult and costly to deploy and use for inference. Minor variations in text inputs, such as typos or misspellings, can cause significant losses in model accuracy. Seeking to improve how these models can be used for *real world* usage and deployments, this thesis focuses on approaches for improving performance by compressing, hardening, and optimizing models' performance based on deployment needs. To explore the challenges with large-scale deployments concerning robustness and inference efficiency, we explore four commonly used language workloads: textual understanding and classification, passage retrieval, and text generation. We chose these broad but connected tasks to ensure that our compression approaches broadly apply to natural language processing. First, we propose a general framework for improving model inference on broad language understanding workloads by studying how unstructured pruning, structured pruning, and quantization can be leveraged to compress models and improve inference speeds. Second, we examine how models can be deployed for usage in web-scale generation and understanding workloads. Leveraging multi-task modeling, asymmetrical pruning, knowledge distillation, and quantization allows for cost-efficient scaling to web-scale workloads. Third, we explore methods of tuning and optimizing dense retrieval methods post-training to ensure they perform well on real-world data. Our experiments yield simple and effective ways of increasing model robustness and decreasing inference costs without any need for retraining or index re-generation. Finally, we discuss future work, focusing on sequential compression approaches to sequence LLMs to allow generative workloads to reach web-scale deployments.

To my wife, Zoe, and my sons, Theodore and Oliver. I would not have finished without you supporting my vision and encouraging me to power through.

ACKNOWLEDGMENTS

This thesis project would not have been possible without the support of many people. Thanks to my adviser, Cheng Xiang Zhai, who helped shape the vision and helped clarify and focus. Also, thanks to my committee members, Alessandro Magnani, Jiawei Han, and Kevin Chang, for their feedback and guidance. Thank you to all the great companies I could work with along the way, Neural Magic, Qualtrics, Walmart, and Neeva. Finally, thanks to my wife, parents, and numerous friends who endured this long process with me, always offering support and love.

TABLE OF CONTENTS

CHAPTER 1: INTRODUCTION

1.1 BACKGROUND

Computers and computational devices have long been improving how humans think and act. At first, using a computer required large rooms of specialized equipment and expert operators. Since then, decades of gradual improvements led to the proliferation of cell phones and virtual supercomputers, which reside in the pockets of billions of people around the globe. These ubiquitous computing devices have become the interface that has brought the world into a truly connected mesh interface. Sharing one's daily experiences with thousands of viewers worldwide with little or no delay is not only possible but as common as driving a car.

In the world of artificial intelligence and natural language processing, there have also been decades of continuous improvement, which have led to models which can do truly impressive feats. Decades ago, it seemed far-fetched that one could ask a system obscure questions and receive the correct answer. Not only can this be done, but questions can come in spoken language, and responses sound natural despite their technological origin.

While personal computing devices can empower incredible experiences, most processing does not happen on devices. Instead, most of the *work* happens in large data centers where economies of scale allow for cost-effective and failure-resistant infrastructure. Using customized deployments, services assemble their experiences by customizing complex workflows and experiences given their customer needs and constraints.

While all data center/cloud workloads are growing, those which support neural network models have grown exponentially. Models have grown from millions of parameters [1] to billions of [2] and will likely reach trillions [3].

Despite the eye-catching growth of the model size, the widespread adoption of these models will likely drive further computational burden. In the earlier part of the 2010s, few companies were training AI models, and even fewer were using them. In 2022 79% of companies had at least 3 models in production, up 17% from a year before [1] [2].

In natural language processing, major advances have been driven by the proliferation of large models. One of the primary goals of natural language processing is to understand human language in all its intricacies and uniqueness. Word representations created by using large textual corpora like GLoVE [4] and Word2Vec [5] allowed for major improvements in tasks

[1]https://www2.deloitte.com/us/en/pages/consulting/articles/state-of-ai-2022.html

[2]https://www2.deloitte.com/us/en/pages/about-deloitte/articles/press-releases/deloitte-state-of-ai-fourth-edition-report.html

like sentiment analysis [6] and question answering [7] [8].
These non-contextual methods provided a simple but effective vector representation, allowing significant language understanding and representation improvements. The field then experienced an explosion of new models and techniques with the arrival of the attention mechanism and large-scale self-supervised pretraining. These two forces collided to create countless large language models such as BERT [9], GPT-2 [10], T5 [11], PALM [12], etc.

1.2 MOTIVATION

The usage of large language models has driven debate about their shortcomings, such as the biases they encode [13] [14], the extent to which they *understand* language [15] [16], and their environmental impact [17] [18]. Despite these challenges, the deployment and usage of these models have been explosive, with thousands of companies deploying unique models optimized to their business needs. Language models are running in search engines like Bing [3]and Google [4], in intelligent assistants like Siri and Alexa [19] and in many specialized use cases all of which are running billions of inference sessions.
The scale of language model deployment has motivated tremendous research into improving the shortcomings. Minor model accuracy and efficiency improvements can lead to millions of dollars in cost savings and empower new usage scenarios that previously seemed impossible. The scale of impact which even minor improvements in inference costs can cause has led to the creation of specialized companies like Neural Magic, Modular AI, OctoML, and DECI, to mention a few. Collectively, they have received hundreds of millions of dollars of investment [5] [6] [7] to explore and commercialize improvements in model inference efficiency.
Large models' accuracy and ability to tackle diverse tasks without specialized understanding have led to treating models like large opaque boxes. Unlike the interpretable and inspectable decision trees they commonly replace, neural models provide little insight into predictions. As a result, large-scale deployments favor treating models as immutable units.

[3]https://azure.microsoft.com/en-us/blog/bing-delivers-its-largest-improvement-in-search-experience-using-azure-gpus/
[4]https://blog.google/products/search/search-language-understanding-bert/
[5]https://deci.ai/news/deci-raises-25m-accelerate-ai-productization/
[6]https://techcrunch.com/2021/11/01/octoml-raises-85m-for-it-for-its-machine-learning-acceleration-platform/
[7]https://neuralmagic.com/blog/neural-magic-series-a/

1.3 THESIS CONTRIBUTIONS

To address the challenges in deploying language models in web-scale workloads, this study studies approaches in compression and augmentation to improve robustness. The contributions of this work vary from highly applied experimental results to broad experiments and methodologies, all of which seek to inform on broadly applicable methods of making language models ready for large-scale production workloads. Specifically, this thesis makes three lines of technical contributions which are explained in detail below.

1.3.1 Introducing and Transferring Sparsity For Efficient Inference

Language models have favored scaling model size because performance improves with scale [20] [21] and large models are more sample efficient [22]. This tendency to favor larger models often means that models are often over-parameterized and can greatly be compressed without large losses in accuracy. Compression methods vary in implementation and practice, but at their core, they seek to decrease the model size or computational cost without sacrificing a larger model's expressiveness. The Lottery Ticker Hypothesis, [23], [24] the finding that some or many sub-networks approximate the original network's performance has helped direct research into the introduction of structured and unstructured sparsity into models. Removing portions of the model greatly increases inference efficiency because the entire network is not needed to be executed simultaneously.

We explore how unstructured sparsity can be used for efficient inference in our work *The Optimal BERT Surgeon* [25] finding that we are able to remove 90% of network weights with little impact to model accuracy. We build on this work in *Sparse*BERT* [26] where we demonstrate that compressed models are to transfer to novel domains and tasks during pretraining without any specialized training or loss in accuracy. We extend this work to novel model types and training regimes in *oBERTa: Improving Sparse Transfer Learning via improved initialization, distillation, and pruning regimes* [27]. Broadly, our work demonstrates how combining unstructured sparsity and quantization with a sparsity-aware serving framework such as DeepSparse [8] or TensorRT [9] model inference can be sped up over 30x. [10].

[8]https://github.com/neuralmagic/deepsparse
[9]https://developer.nvidia.com/tensorrt
[10]https://neuralmagic.com/blog/obert/

1.3.2 Robust and Efficient Semantic Retrieval

Using bi-encoders and vector-based representations of documents and queries has led to major advances in information retrieval [28]. Leveraging language models as vector representation has led to effective and scalable semantic search, which can be applied to e-commerce [29] [30], question answering [31], and web search [32].
Despite the retrieval ability of bi-encoder usage in the real world can be difficult because of their sensitivity to typos and noise [33] and inherited difficulty in scaling workload as they are based on compute-hungry transformers. We study how bi-encoder models perform with noisy queries in our work *Noise-Robust Dense Retrieval via Contrastive Alignment Post Training* [34]. Contrastive Alignment Post Training, we can reduce accuracy losses on queries with typos by 55% without model retraining nor index regeneration.
Building on the simplicity of post-training modular optimization, we explore how to improve inference efficiency in *Quick Dense Retrievers Consume KALE: Post Training Kullback–Leibler Alignment of Embeddings for Asymmetrical dual encoders* [35]. Leveraging post-training compression and representation alignment, we demonstrate that it is possible to improve inference efficiency by over 4x with only minor losses in retrieval accuracy.

1.3.3 Scaling Multi-Lingual Classification and Abstractive Summarization to Web-Scale workloads

While advances in language model quality have driven wide adoption for production workflows scaling the models to web-scale workload can prove difficult. Even minor compression improvements can save millions of dollars when scaling to consistent large inference workloads associated with web-scale usage.
The use of sequence-to-sequence models has led to massive improvement in machine translation [36], abstractive summarization [37] and speech/audio transcription [38]. Part of the success of these models is driven by their ability to map an input to an output despite variability in length, type, or even domain. While effective, these models carry a high computational load as their architecture can be full of inefficiencies. While the encoder portion of the model runs once on the input, the decoder produces outputs iteratively until the end of the input tag is produced. As a result, usage can become bottle-necked when decoders produce long inputs, and performing batch processing results in non-optimal computing usage as outputs have differing lengths.
In *To Asymmetry and Beyond: Structured Pruning of Sequence to Sequence Models for Improved Inference Efficiency* [39], we explore the interplay between scale, model symmetry, pruning, and inference efficiency. Using T5 models on the XSUM [40] and CNN/DailyMail

[41], we show evidence that a deep encoder and shallow decoder is optimal and can lead to nearly 3x speedups in inference efficiency with less than 2 points lost in ROUGE-2.
In language understanding and text classification, traditional language models like BERT [9] are monolingual by design and focus on being used only for the language they were trained in. While using machine translation as a processing step can provide effective predictions [42], this approach requires additional inference and effective machine translation between all languages. Multi-Lingual language modeling seeks to avoid the difficulties of mass translation or many monolingual language models by simultaneously training a representation for many languages. This approach has widespread usage and is used for broad language agnostic question-answering classification and generation.
Using language models has become a natural part of the text-understanding toolkit for companies focusing on understanding customer insight. At Qualtrics, workloads exist for extracting insights from customer feedback, such as sentiment, emotion, topics, and actionability. While multi-lingual language models such as XLM-R [43] allow classification workloads a simple and effective path to multi-linguality, their size can make web-scale deployments expensive and difficult.
In our work *Compressing Cross-Lingual Multi-Task Models at Qualtrics* [44], we study how the use of quantization, multi-task learning, and knowledge distillation to improve model performance by 15x with minor losses in accuracy. In this work, we highlight the impact of leveraging knowledge distillation during fine-tuning and the importance of distilling from task-optimized teachers.

1.4 DOCUMENT STRUCTURE OVERVIEW

In chapter 2, we provide a literature review that covers language models, tasks and methods of using language models, methods of model compression, and shortcomings where language models can be brittle or produce unwanted outputs. In chapter 3, we discuss some of the work we have completed around leveraging unstructured sparsity, knowledge distillation, and quantization to train, compress and transfer efficient inference models. In chapter 5, we introduce and discuss our work improving the inference efficiency of multi-lingual text classification using quantization, structured pruning, multi-task modeling, and task-specific knowledge distillation. In chapter 4, we discuss our work on improving the efficiency and robustness of bi-encoder-based retrieval using post-training alignment and compression. Finally, in chapter 6, we discuss the role of scale in compression, provide suggestions on model sizing and possible gains from compression approaches, and discuss future work.

CHAPTER 2: LITERATURE REVIEW

2.1 OVERVIEW

In this chapter, we provide a broad overview of the relevant subject which we will discuss. First, we discuss language models, their importance, and their limitations. Next, we discuss the broad field of model compression and some popular approaches for model compression. Third, we discuss the application of language models as applied to information retrieval.

2.2 LANGUAGE MODELS

2.2.1 What Is Language Modeling and Why Is It Useful

Language modeling is a method of assigning a probability distribution over some form of language, like input. When applied to tokens in spoken or written human language modeling models, the probability of a token w_i given the previous i tokens as shown in equation 2.1:

$$P(w_1, \ldots, w_m) = \prod_{i=1}^{m} P(w_i \mid w_1, \ldots, w_{i-1}) \approx \prod_{i=1}^{m} P(w_i \mid w_{i-(n-1)}, \ldots, w_{i-1}) \tag{2.1}$$

Language models (LM) can be useful methods to represent natural language because they allow models to differentiate the meanings of sentences based on context. In other words, a model can understand that the word 'fly' can mean different things in the sentences: 'You look fly,' 'Let's fly away!', 'That is a fly.

While language modeling is not a new concept, it was not until the introduction of Neural Network-based LM that these representations could serve as general understanding frameworks. Before these Neural Network Language Models (NNLM), most language modeling usually focused on modeling some form of an N-gram where the probability of a word only depends on the previous N-word. Large Neural-Network-based LMs are the first step in a Natural Language Processing (NLP) application as a way of turning some form of textual input into a representation in a vector space.

Language models are created using many training objectives, but general models tend to be either auto-encoding (AE), auto-regressive (AR), or some combination. AR models like Elmo [45] or GPT-2 [10] learn an LM by predicting the next token in a sequence. AE models like BERT [9] and ELECTRA [46] learn an LM by reconstructing some sequence portion.

2.2.2 Transformers

Language Models are commonly built using multiple transformer layers to capture long-term input dependencies using self-attention [36]. Each transformer usually has some variation of two sub-components: *multi head attention (MHA)* and *fully connected feed-forward networks (FFN)*. MHA contains many self-attention heads, each of which has three sub-components: queries (**Q**), keys (**K**), and values (**V**). The output of the attention component is the concatenation of each attention head and is fed into the FFN. The attention of each *head* in MHA is formulated as:

$$Attention(Q, K, V) = softmax\left(\frac{QK}{\sqrt{d}}\right)V, \tag{2.2}$$

where d is the dimensionality of **K** and is used as a scaling parameter. The FFN is a fully-connected feed-forward network with linear transformations and an activation function such as ReLU or GeLU.

2.2.3 BERT

Building on the success of ELMo, leveraging the transformer architecture [36], and taking the learning from other contextual word embeddings [47] [48] Devlin et al., 2018 introduced BERT, which stands for Bidirectional Encoder Representations from Transformers. BERT is an AE LM that uses modified stacked Transformer encoders (12 layers for a small model and 24 for a large model) to build a contextual language representation. Instead of using character-level convolutions or fixed word vectors as a starting point, BERT leverages a piecewise tokenization [49], which sets a vocabulary size of 30,000.
Like other language models, BERT trains using unsupervised pre-training on a large text corpus. Unlike previous models, BERT introduces two new training objectives to steer the model: Masked Language Modeling (MLM) and next sentence prediction (NSP).
MLM reformulates language understanding as a cloze task [50], where the model's goal is to predict what a hidden word in a sentence may be. To train using MLM, BERT introduces a new token $[MASK]$ to represent the hidden word. 15% of each of the corpus tokens are selected to be replaced, of which 80% (12% of the corpus) is replaced with $[MASK]$, 10%(1.5% of the corpus) is replaced with a random token, and the remaining 10% are left alone. The model predicts the word when it finds a $[MASK]$ token. NSP is a training method inspired by question-answering (QA) systems, which tend to have two sentences to reason on a query and a context passage. In NSP, the model is fed text, which combines two sentences, A and B, with the unique separation token [SEP]. In 50% of the NSP samples, sentence B

directly follows A, while in the remaining 50%, A and B are selected randomly. The model has a binary training goal if the sentences are next to each other in the original text.

2.2.4 Beyond BERT

Besides BERT and Elmo, there has been considerable research into additional language models. RoBERTa [51] improves on BERT by training on a larger corpus for a longer time. XLNET [52] combines AE and AR while avoiding some of the pitfalls of each method by modifying AR to maximize the expected log-likelihood of a sequence concerning all permutations of factorization order. XLNET also removes the notion of a $[MASK]$ token to avoid training the model with a token that never occurs in text and implements the whole architecture using the Transformer-XL [53]. ALBERT [54] explores the role of size in LM, finding that parameter weights can be shared across layers meaning they can have 18 times fewer parameters and train 1.7x faster than regular BERT all while producing similar language representation to BERT. DistilBERT [55] creates a smaller LM using knowledge distillation resulting in a similar performance to BERT with a 40% smaller model. GPT [48], GPT-2 [10], and GPT-3 [2] build an AR LM more suited toward language generation by using progressively larger models and a modified transformer decoder architecture. ELECTRA [46] produces a model with comparable performance to BERT with substantially shorter training by having the model predict all tokens in a sentence instead of the $[MASK]$ token and by corrupting the input using a Generator similar to that of a GAN. Beyond these few models, we mention countless other optimizations and applications of this large-scale NNLM.

2.2.5 Relation To Thesis

Language models have become cornerstones of most approaches to understanding language. Driven by this, in this thesis, we mainly study BERT, X-LMR, and T5 [11] as they are broadly used. While more modern models have improved accuracy and efficiency, these models are not used nearly as often. Our work focuses on improving performance for production deployments' *workhorses.*

2.3 MODEL COMPRESSION

Given the ability to classify, predict and generate insights, AI models have become a large and common form of the computational workload. While these models can gener-

ate impressive results, using them at scale can be expensive and difficult. They commonly require specialized inference infrastructures such as Graphics Processing Units (GPU)s or Field Programmable Gate Arrays (FPGA)s [56]. The high cost of using models has driven research to explore how to decrease the model size without losing accuracy.
While many successful compression approaches have been pioneered outside of natural language processing (NLP)[57][58] [59], Transformer models can be fragile[60], as minor perturbations can lead to model collapse. Compression approaches usually start from a set of trained parameters θ, e.g., a Transformer-based language model, and aim to produce a different model θ^* which approximates the accuracy of θ w.r.t. a given task-dependent loss function $\mathcal{L}$ while minimizing its cost c, but at lower parameter count and increased inference efficiency. In this formulation, model compression becomes an optimization of $((\mathcal{L}_{\theta*} - \mathcal{L}_{\theta}) + c(\theta*))$. Models are compressed by reducing the size or computational complexity of execution so that their performance mirrors or approximates that of the original model [61].

2.3.1 Iterative Compression

Compression schemes are often motivated by weight saliency metrics which minimize the loss in accuracy due to the failure in the expressivity of the network. While there has been some success in compressing models without retraining [62] [63] or using a single compression step [64], it is more common to compress models in a gradual iterative fashion. In this paradigm, a network is compressed in stages. Compression is applied at each compression step, where some portion of the network is selected for reduction, and the network is further trained to recover its complete accuracy.

2.3.2 Pruning

Pruning is a set of compression approaches that decrease the model size and improve execution cost by removing network portions [58].
Unstructured pruning removes individual neurons by setting them to zero [57], which can be exploited for storage and computational speedup. Unstructured Pruning seeks to find a proxy of importance to identify what portions of the network can be removed with the smallest impact on accuracy [58]. Zeroth order methods such as magnitude pruning [57], [65] assume weights as a proxy for importance and the smallest weight. First-order methods such as movement pruning [66] estimate importance by measuring the movement of weights once they are transferred to a new task where weights that do not move are considered less

important.
Second-order [58], [67], [68] methods such as Optimal Brain Surgeon (OBS) leverage complex Hessian approximations to determine the impact which Pruning may have. Other work has focused on creating pruned networks by identifying sparse *lottery tickets* networks which transfer well to downstream tasks and approximate the uncompressed model [23], [24].
In the realm of transformer-based LLMs, it has been shown that models can be compressed during pre-training so that there is little to no loss in accuracy when fine-tuned [69]. While unstructured Pruning can lead to massive compression in model parameters, improving inference speeds requires specialized hardware or sparsity-aware inference engines in practice.
Structured pruning [58] removes entire structural portions of a network, which in the scope of transformer-based language models layer in the encoder/decoder, decreasing the hidden size of smaller structures like attention heads. Structured pruning approaches require a structural understanding of the model to be successful. For transformer-based language models, attention heads vary in importance, and nearly 40% of heads can be removed without major impact on accuracy[70], [71]. Existing research has focused on evaluating how many transformer layers can be removed [72] and the order in which they can be removed [73]. Models like BORT [74] combine structured Pruning with an optimization approach to produce smaller models designed around theoretically optimal sizes.

Semi-structured pruning is an intermediate approach where portions of the model are removed with a small, consistent grouping, such as a rectangular weight grouping [75] by setting their weights to zero. This approach is set to zero. This approach has recently gained popularity thanks to efficient computational support in GPUs leading to wide-scale, measurable inference improvements.

2.3.3 Knowledge Distillation

Knowledge Distillation (KD) [76] is an approach in which a compressed *student* model is trained not to match the outputs of the dataset but the outputs of a larger and more accurate *teacher* model by adding a loss component which minimizes the Kullback–Leibler divergence between the two output distributions as shown in Equation 2.3. KD is a form of label softening as a traditional target is a *hard* one-hot vector representing the correct class. The learned outputs represent the candidate label distribution of a well-trained model. uses a hardness parameter to control the mixture of regular loss and distillation loss and a temperature parameter to control the softness of the probability distribution.

$$\mathcal{L} = h\mathcal{L}_d + (1-h)\mathcal{L}_\ell. \tag{2.3}$$

$$\mathcal{L}_d = D_{KL}(\theta^* \parallel \theta^t) = \sum_{x \in \mathcal{X}} P(x) \log\left(\frac{P(x)}{Q(x)}\right). \tag{2.4}$$

KD has been broadly applied to transformer-based language models leading to general-purpose compressed models like DistillBERT [55], while TinyBERT [77], MobileBERT [78], and MiniLM [79] have the student approximate the teacher's intermediate representations, obtaining better results at the cost of higher complexity.

2.3.4 Quantization

Quantization decreases the cost of inference and model size by lowering the precision of weight and activation within a model [80]. Most networks have their weights represented using *int32*. Quantization methods produce θ^* by representing weights using less precise data structures such as *int16* and *int8*. While quantizing without retraining the model is possible, minor rounding errors can lead to significant losses in accuracy. As a result, performing Quantization-Aware Training (QAT) is a common approach to ensure minimal loss in accuracy in compressed models. Specifically, QAT simulates the rounding effects on floating point values and leverages a Straight-Through Estimator (STE) to approximate the gradient, as quantization operations are not differentiable. While it is possible to produce networks that use one, two, or three bits for weight representation, lack of operator support can make it challenging to realize speedups, so most quantization focuses on int8 and int4. Q8BERT [81] can apply QAT and improved training regimes to produce a simple and extensible quantized language model. At the same time, TernaryBERT [82] uses a complex distillation-based approach to obtain highly low-bit representations. In contrast, Fan et al. '20 [83] propose a scheme that randomly quantizes group weights during training, leading to more accurate compressed models.

2.3.5 Relation To Thesis

This thesis primarily leverages unstructured pruning, structured pruning, quantization, and knowledge distillation. We focus our work on these compression approaches as they are widely deployed, and their usage provides the most direct path to realize inference speedups using standard hardware. Using these approaches, we seek to improve model inference speeds using commodity-grade central processing units (CPU) and graphics processing units

(GPU).
This thesis explores the novel application of these methods for information retrieval, text classification, and text generation in ways not previously explored. Additionally, we explore how these compression approaches work on novel model architectures and how they can be used in conjunction with maximal compression. Our work introduces a wide range of experimental findings about optimal methods for improving robustness and inference efficiency for existing and novel workloads.

2.4 NEURAL METHODS FOR INFORMATION RETRIEVAL

The field of Information Retrieval has long studied how best to retrieve the most relevant information given constraints in corpora, inference needs, and many task-specific conditions. While term-based methods have long driven retrieval, the growth of datasets and language models has seen surging popularity both in research and real-world deployments.

2.4.1 Bi-Encoders

Bi-Encoders, commonly called dual-encoders or dense retrievers, decompose ranking by leveraging the inner product of query and document representations to produce a relevance score for query document pairs. As their name suggests, bi-encoders leverage two encoders that run independently, one for the query and one for the passage. They are commonly called dense retrievers because of the density of their representation vectors compared to the sparsity of term-based retrieval methods. Dense retrievers can leverage contextual word representations without massive computational overhead by producing query and document representations. Since their document representations are query invariant, they can be precomputed and loaded into an Approximate Nearest Neighbor (ANN) such as FAISS [84]. The k closest documents can be found for each query with minimal latency at run time. Bi-encoders leverage LLM such as BERT [9] for their text representation. As a result, they are often limited to ranking short passages of text and are commonly referred to as Dense Passage Retrievers (DPR) [28] [85]. Driven by their efficiency in deployment and relevance performance, DPR-based models have rapidly become the building blocks for systems doing product search [29], open domain question answering [28] and customer support [86].
Recent work has heavily focused on improving the relevance of DPR models by improving the negative sampling using methods like ANCE [32] and in-batch negatives [87]. While effective DPR models are brittle to shifts in the domain, minor variations can cause a complete collapse in relevance. Li et al. '2022 introduced methods for improving such performance

by having a single query encoder leverage multiple document encoders to transfer between domains [88]. While effective, such a method carries a high computational load as multiple indexes must be maintained and updated.

2.4.2 Cross-Encoders

Cross-Encoders are an application of language models which generate a rank or re-rank documents by producing a relevance score by scoring each possible query document pair. This method formulates the task as a binary classification where the query and candidate document into one text input, and a language model will predict whether this pair is relevant. A Cross-Encoder does not produce a sentence embedding and, as a result, can be much more computationally expensive when compared to a bi-encoder. While less efficient, Cross-Encoder performs better than Bi-Encoders and is more robust to noise or domain shift. [89].

2.4.3 Training Methods

Data Augmentation (DA) is a popular approach for improving how well models perform on new or noisy data. In data augmentation, training is extended by augmenting the training data with modifications or perturbations which match the desired model behavior. DA is extremely common in computer vision where training data is commonly rotated, blurred, cropped, or zoomed-in/out [90] [91]. DA has become increasingly more popular in NLP and has been used to improve model performance [77], simulate large-scale training data when it is not available [92], and mitigate bias [93] in existing datasets. A detailed survey on DA approaches for NLP has been complied by Feng et al. 21' [94].
Contrastive Learning (CL) builds on the notion of a contrastive loss [95], which seeks to create clusters in the embedding space such that examples with a shared class are far from other classes but close to each other. Much like learning that queries with noise have a shared intent, Schroff et al. 15' leverage contrastive learning to recognize faces despite different angles and perspectives [96] using a triplet loss. This approach is a natural fit for the world of search as relevance is at its core clustering relevant items close together and far from irrelevant items. Recently, contrastive learning has become a method for learning relevance at the corpora scale [32] and improving DPR on noisy queries, [33] [97].

2.4.4 Compressing Bi-encoders

The widespread utility of bi-encoders has demonstrated the need for compression. Seeking to improve the size of vector-based indexes has led to binary compression of dense representations [98] and broad experimentation about reducing the end-to-end size of retrieval systems [99]. While effective in the domain, these approaches have drawbacks as learned compression approaches can be brittle to domain shifts [100]. Choi et al. '21 [101] show that using knowledge distillation with multiple teachers can maximize the performance of dense retrievers.

2.4.5 Relation To Thesis

This thesis focuses on the bi-encoder models, which use language models to generate their vector representations. Our focus on these models is driven by the growing interest in using vector-based retrieval models to provide context for other language models. Methods that leverage bi-encoders have quickly gone from specialized deployments to being the common building block for many research projects. Despite their broad adoption as a retrieval layer for research studies improving efficiency and robustness is more nascent. In subsequent chapters, we focus on finding and mitigating weaknesses in noisy inputs and inference efficiency.

2.5 SEQUENCE TO SEQUENCE MODELING

2.5.1 Scaling Laws

Scaling Laws has become an increasingly important area of study as models' size and training data grows. Performance of the transformer-based language model improves with the relation to model size [48] and that larger models outperform smaller models [2] on most NLP tasks. Increasing the training corpus size can lead to large improvements in performance, and model sizes can have a *optimal* training data size [102]. Li et al. (2020) [103] explore the relationship between model size and training efficiency finding larger models train faster and are more robust to pruning and quantization [104].
Rosenfeld et al. 2020 demonstrate that unstructured pruning impacts follow scaling laws [105] where larger models can be pruned with greater ease. Despite this broad study of scaling laws, to our knowledge, we have not found any research focusing on the scaling laws of sequence-to-sequence models for summarization tasks.

2.5.2 Asymmetrical Sequence To Sequence Modeling

broadly refers to non-uniformity between encoder and decoder model shape or attributes. Training and inference procedures should match as closely as possible [106] [107] as improvements in training loss during optimization result in improvements in model performance during Inference. While this may lead to the best model performance, it ignores the variable inference cost of models sequence to sequence models.
During Inference, latency is dominated by the asymmetric execution of the language model. The auto-encoding encoder executes once over the entire input sequence, while the auto-regressive decoder executes iteratively until an end-of-sequence token is produced.
Kasai et al. demonstrated how the sequence-to-sequence language model performance for machine translation is dominated by the encoder depth [108]. Tay et al. 2021 extend this work by finding a *DeepNarrow* which shows that for broad language modeling, it is possible to have 50% fewer parameters and a 40% faster inference with no loss in accuracy [109].

2.5.3 Compressing Sequence To Sequence

Compressing Sequence-to-sequence is a growing area of study where approaches from regular, efficient inference has shown some transfer ability. Shleifer et al. show that it is possible to gain 1.93x speedup on a BART summarization model by applying structural pruning [110] but find compression approaches differ in their success depending on the dataset. Leveraging semi-structured pruning, Lagunas et al. can gain a 1.19 speedup [111] for minor losses in accuracy. While they find that the encoder is easier to prune than the decoder, they do not use this evidence of asymmetry to speed up performance further.
Li et al. investigate how to enable quantization, finding that without specialized distillation during quantization, performance collapses [112]. Leveraging that generation occurs iteratively, and some tokens are easier to generate than other CALM [113] apply early exiting to improve inference speed by 1.4x. While existing work has found interest in asymmetry, it has not been studied directly, nor has relationships in model scale been explored.
While there are other approaches such as knowledge distillation [76] [114] [77], quantization [81], early exiting [115] and token pruning [116] we focus our work on the impact structural pruning and its relation to symmetry in size of sequence to sequence models. We focus on this form of compression as we found applications of asymmetric pruning [110] [112] without

any direct study of the impact of variations in symmetry.

2.5.4 Relation To Thesis

This thesis focuses on how sequence-to-sequence modeling can be scaled to web-scale workloads by improving their inference efficiency. Sequence-to-sequence models have become a common building block for NLP workloads. They can map inputs and outputs with varying lengths and modalities, providing a powerful architecture for conditional generation. In studying this application, we leverage model scaling laws, asymmetrical modeling approaches, and existing compression approaches.

CHAPTER 3: INTRODUCING AND TRANSFERRING SPARSITY FOR EFFICIENT AUTO-ENCODER INFERENCE

3.1 OVERVIEW

For the last decade, the common paradigm in using deep learning has been to improve model performance by improving architecture and scale. These models feature larger and larger, highly interconnected layers, otherwise referred to as dense. While this approach has continuously led to improvements in model performance, it is not without drawbacks. In 2011 a state-of-the-art computer vision model could run using a laptop. Now running a state-of-the-art language mode requires a cluster of specialized GPUs, which can cost upwards of 100,000 dollars an hour and draw kilo-watts of power.
Inspired by the sparsity of the connections of neurons in the brain, unstructured sparsity seeks to improve the model efficiency by turning densely connected models into sparse models, which as a result, are far more efficient. While a large portion of existing research has focused on theory and high-level implementations, our work is focused on using the same sparsity to realize true measurable inference speedups.
This chapter will discuss our broad experimentation focused on leveraging unstructured sparsity to improve efficiency for language model inference. First, our work examines *The Optimal BERT Surgeon* where optimal zeroth and second-order pruning approaches are combined with quantization and structural pruning for a systematic approach for improving inference efficiency without using GPUs. Next, our work discusses *Sparse*BERT* and broad experimentation on how sparse language models can be transferred to novel domains and tasks without further optimization. Finally, our work discusses oBERTa, which extends earlier experiments in sparse language modeling while improving training methods, model initialization, and distillation to deliver compelling inferences for overall text classification workloads.

3.2 THE OPTIMAL BERT SURGEON: SCALABLE AND ACCURATE SECOND-ORDER PRUNING FOR LARGE LANGUAGE MODELS

3.2.1 Overview

In this section, we consider the problem of sparsifying BERT models, a crucial building block for natural language processing, to reduce their storage and computational cost. We introduce the *Optimal BERT Surgeon* (oBERT), an efficient and accurate pruning method based on approximate second-order information, which we show to yield state-of-the-art results for compression in both stages of language tasks: pre-training and fine-tuning. Specifically, oBERT extends existing work on second-order pruning by allowing for pruning blocks of weights and is the first such method applicable to the BERT scale. Second, we investigate compounding compression approaches to obtain highly compressed but accurate models for deployment on edge devices. These models significantly push the boundaries of the current state-of-the-art sparse BERT models concerning all metrics: model size, inference speed, and task accuracy. For example, relative to the dense BERT, we obtain 10x model size compression with ¡ 1% accuracy drop, 10x CPU-inference speedup with ¡ 2% accuracy drop, and 29x CPU-inference speedup with ¡ 7.5% accuracy decline.

3.2.2 Introduction

Pre-trained Transformer models [36] [9] robust language representations which can be specialized on various tasks. Given their massive growth [10] [117], techniques for reducing their computational overheads have become popular. One classic technique is Knowledge Distillation (KD) [76], which transfers knowledge from a larger teacher to a smaller student model. Other work has leveraged lower-precision representations to produce quantized models. Our primary focus is an orthogonal approach to applying unstructured and block pruning, i.e., removing individual weights, to produce compressed but accurate language models. Figure 3.1 provides a comparative overview of state-of-the-art results for unstructured pruning.

This section introduces a method for improved unstructured and semi-structured (block) pruning by leveraging the second-order approach pioneered by the Optimal Brain Surgeon framework [58], [67], which we scale for the first time to LLMs. Further, we put our results in the context of a compound compression approach, which combines several compression techniques to obtain sparse models which we execute on a sparsity-aware CPU-based runtime [119], showing order-of-magnitude speedups at low accuracy loss. In summary, our contributions are as follows:

- We thoroughly explore weight pruning approaches applied to LLMs, including lottery tickets, movement pruning, magnitude, and second-order pruning.
- We introduce a general second-order pruning method called *Optimal BERT Surgeon* (oBERT), which supports unstructured and block pruning and is the first second-order method to be both highly accurate and scalable to the dimensionality of BERT models.
- We illustrate the benefits of oBERT by significantly improving upon existing state-of-the-art pruning methods in both stages of language tasks: pre-training and fine-tuning.

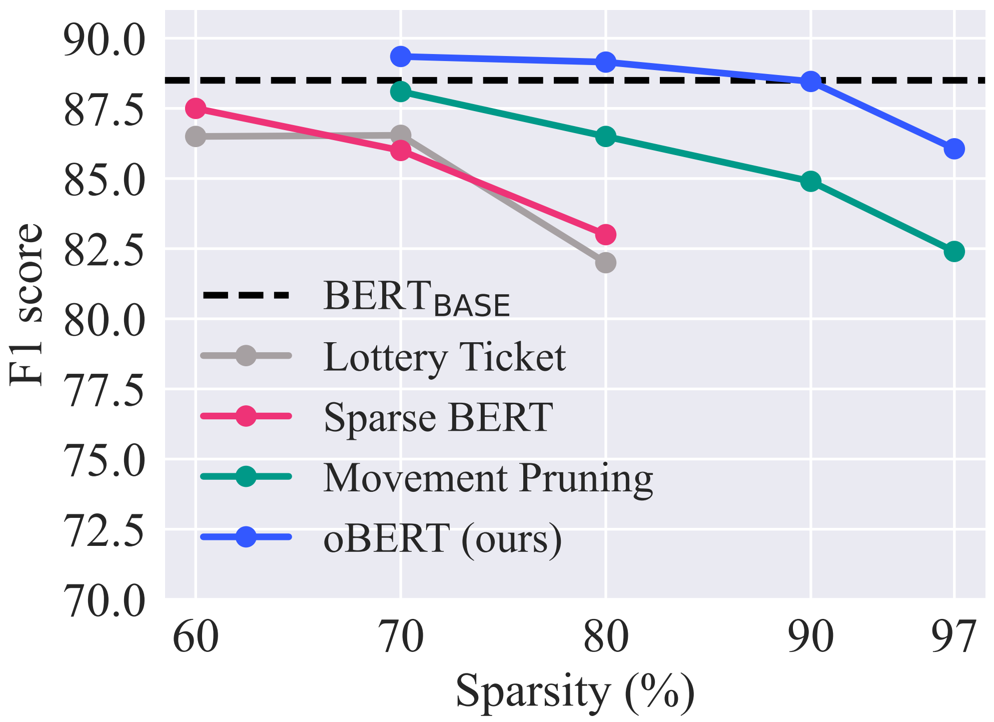

Figure 3.1: Performance overview relative to state-of-the-art unstructured downstream pruning methods [24], [118], [66], in this order, of the BERT-base model on the SQuADv1.1 task.

For illustration, when pruning BERT, oBERT outperforms Movement Pruning (MvP), the most accurate prior approach, by more than 2% absolute F1 score at the same sparsity and can match the accuracy of MvP models with 3x fewer parameters.

- We investigate the applicability of this pruning method in a framework which *compounds* popular compression approaches for LLMs, i.e., applying pruning in combination with layer dropping and quantization. In this context, we show that our resulting sparse models provide order-of-magnitude improvements compared to other compound compressed models and can be easily deployed for CPU inference.

3.2.3 Background and Related Work

Transformer LLMs are usually built using multiple transformer layers with self-attention [36]. Each transformer has a variation of two sub-components: multi-head attention (MHA) and a fully connected feed-forward network (FFN). Given the massive size of well-performing models, there has been growing interest in LLM compression. They are fragile as minor perturbations can lead to model collapse [60]. Pruning schemes are motivated by weight saliency metrics representing the loss in accuracy due to pruning. It is common to prune in iterative steps, each removing weights until a desired sparsity level is reached. Now, we briefly overview existing approaches.

Structured pruning for LLMs focuses on reducing the number of layers and attention heads and requires a structural understanding of the model. [70] and [71] demonstrated that for some tasks, nearly 40% of attention heads could be removed without a major impact on accuracy. Other work has focused on removing layers [72] and the order in which they are removed [73]. In some of our experiments, we apply standard "direct" layer dropping with pruning.

Semi-structured pruning is an intermediate approach by which smaller groups, e.g., rectangular sets of weights [75], are set to zero. This approach has recently gained popularity thanks to efficient computational support. We extend the second-order pruning formulation to such groupings and show results for a specific grouping supported by a CPU-inference engine.

Unstructured pruning removes individual weights by setting them to zero. Gradual Magnitude Pruning (GMP) is a classic approach, which makes use of weight magnitudes as a saliency metric for pruning [57], [65].

First-order pruning methods use a gradient-based formulation of the saliency metric. A popular method is **Movement Pruning (MvP)** [66], specifically designed for pruning in the fine-tuning stage. Intuitively, it removes weights that are moving toward zero. The resulting models were the first to achieve high sparsity with tolerable accuracy loss. Before our work, this approach set state-of-the-art results for unstructured pruning.

Second-order pruning methods [58], [67], [68], [120] were developed in the context of image classification, and leverage complex approximations of the loss curvature. However, second-order pruning methods require an approximation of the inverse Hessian, which is expensive to store and compute for LLM parameter counts. Our proposed approach is similar to WoodFisher/M-FAC methods [68], [120], but is the first to work accurately at the LLM scale. Specifically, the Woodfisher approach is infeasible at the BERT scale, as it requires storing gradients for inverse Fisher calculation in memory at the point of pruning. The M-FAC approach scales, but we show that its parametrization yields worse pruning results. This is because M-FAC performs full-matrix (non-blocked) inversion by default, which is inherently noisy. In addition, we extend the theoretical OBS approach to semi-structured (block) compression. We also show that our method can be applied during LLM pre-training and fine-tuning, yielding state-of-the-art results in both regimes.

Knowledge Distillation [76] trains a smaller student model against outputs of a larger teacher model by adding a loss component that minimizes the KL-divergence between the two output distributions, which is the approach we adopt in our setup too. A hardness parameter is used to control the mixture of regular and distillation loss, and a temperature parameter is used to control the softness of the distribution. Contrary to this, approaches like DistilBERT [55], TinyBERT [77], MobileBERT [78], and MiniLM [79] utilize more complex distillation schemes, based on transferring knowledge from intermediate model's representations. Our sparse models provide order-of-magnitude improvements upon some of these methods.

Quantization represents weights and activations in lower precision [80], and was used to obtain models such as Q8BERT [81] and TernaryBERT [82].

[121] uses information about the Hessian spectrum to choose quantization bit-widths, whereas [122] approximates the Hessian trace for structured pruning. These Hessian-based approaches differ from the one we propose, as we use entirely different inverse-Hessian approximations to guide pruning decisions. Our work focuses on *weight pruning* and computational speedups achievable on commodity CPUs. As such, the methods we investigate are orthogonal to quantization. Moreover, it is impossible to directly compare to low-bandwidth quantized models as most inference frameworks do not support such custom formats. Therefore, we will only use the standard Quantization-Aware Training (QAT) to 8-bit weights, which is well-supported on Intel CPUs, and showcase the resulting speedups in conjunction with layer dropping and weight pruning.

Downstream compression methods attempt to compress directly while fine-tuning a specific task. The MvP method is specially designed for this setup. **Upstream compression** methods compress during the pre-training phase, reducing the need for task-specific pruning. [24] examined the "Lottery Ticket" strategies [23], which, as we illustrate later, incur huge accuracy loss even at moderate sparsities. Recent work "Prune Once for All" (Prune OFA) by [69] showed that well-tuned magnitude pruning could be competitive with downstream methods like MvP.

We first examine the performance of prior pruning methods, notably MvP, Prune OFA, and Lottery Tickets, relative to the new second-order oBERT method. The approach we propose consistently improves upon all these prior methods, both in the pre-training (upstream) and fine-tuning (downstream) stages, and can be compounded with other compression techniques to obtain models that are smaller, faster, and more accurate than models like DistilBERT, TinyBERT, and block MvP.

3.2.4 The Optimal BERT Surgeon (oBERT)

3.2.5 Generalized Second-Order Block Pruning

The pruning problem starts from a well-optimized dense model $\mathbf{w}^* \in \mathbb{R}^d$, and aims to find a sparse version of $\mathbf{w}^*$, where many of the weights are set to zero, and the remaining weights may be updated accordingly to preserve the loss. It is common for this process to occur gradually, i.e., by progressively removing the weights. A classic approach [58], [67] for "optimal" pruning of weights from $\mathbf{w}^*$ at a step is to expand the loss function $\mathcal{L}$ locally around $\mathbf{w}^*$ for a sparse 0/1 weight mask $\mathbf{M}$. If we denote by $\mathbf{w}_M = (\mathbf{M} \odot \mathbf{w}^*)$, the model resulting from the Hadamard (element-wise) product between $\mathbf{M} \in \{0,1\}^d$ and $\mathbf{w}^*$, we can use the Taylor expansion at $\mathbf{w}_M$ to obtain:

$$\mathcal{L}(\mathbf{w}_M) \simeq \mathcal{L}(\mathbf{w}^*) + (\mathbf{w}_M - \mathbf{w}^*)^\top \nabla \mathcal{L}(\mathbf{w}^*) + \frac{1}{2}(\mathbf{w}_M - \mathbf{w}^*)^\top \mathbf{H}_{\mathcal{L}}(\mathbf{w}^*)(\mathbf{w}_M - \mathbf{w}^*). \tag{3.1}$$

Given that $\mathbf{w}^*$ is well-optimized, it is reasonable in practice to assume that $\nabla \mathcal{L}(\mathbf{w}^*) \approx \mathbf{0}$ [68]. Then, the change in loss incurred by pruning a subset of weights can be expressed as

$$\delta \mathcal{L}(\delta \mathbf{w}) \simeq \frac{1}{2} \delta \mathbf{w}^\top \mathbf{H}_{\mathcal{L}}(\mathbf{w}^*) \delta \mathbf{w} \tag{3.2}$$

where $\delta \mathcal{L}(\delta \mathbf{w}) \mathcal{L}(\mathbf{w}_M) - \mathcal{L}(\mathbf{w}^*)$ and $\delta \mathbf{w} \mathbf{w}_M - \mathbf{w}^*$. A popular way of approximating the Hessian at $\mathbf{w}^*$ is via a dampened empirical Fisher information matrix [67]:

$$\mathbf{H}_{\mathcal{L}}(\mathbf{w}) \simeq \widehat{\mathbf{F}}(\mathbf{w}) = \lambda \mathbf{I}_d + \frac{1}{m} \sum_{i=1}^{m} \nabla \mathcal{L}_i(\mathbf{w}) \nabla \mathcal{L}_i^\top(\mathbf{w}) \tag{3.3}$$

where $\lambda \geq 0$ is a small dampening constant, $\mathbf{I}_d \in \mathbb{R}^{d \times d}$ identity matrix and m is the number of gradient outer products used to approximate the Hessian. Given the positive-definiteness of (3.3), the quadratic form (3.2) is always nonnegative, which is why we will refer to $\delta \mathcal{L}(\delta \mathbf{w})$ as a *loss increase* incurred by pruning.

Returning to our pruning problem, we assume we wish to identify a block of weights Q of a given shape whose removal by zero-masking would incur a minimum increase in loss. This leads to the following constrained optimization problem:

$$\begin{aligned} \min_{\delta \mathbf{w}} \quad & \frac{1}{2} \delta \mathbf{w}^\top \widehat{\mathbf{F}}(\mathbf{w}^*) \delta \mathbf{w} \\ \text{s.t.} \quad & \mathbf{e}_k^\top \delta \mathbf{w} + w_k = 0, \quad \forall k \in \mathrm{Q} \end{aligned} \tag{3.4}$$

where $\mathbf{e}_k \in \mathbb{R}^d$ stands for the k-th canonical basis vector. This derivation is known [67];

however, the optimal pruning update has only been derived for pruning individual weights. Here, we will provide a generalized solution, which applies to general Q. First, for convenience, we express the system of $|\mathrm{Q}|$ equality constraints in matrix-equation form as $\mathbf{E}_{\mathrm{Q}}\delta\mathbf{w} + \mathbf{E}_{\mathrm{Q}}\mathbf{w}^* = \mathbf{0}$, where $\mathbf{E}_{\mathrm{Q}} \in \mathbb{R}^{|\mathrm{Q}| \times d}$ is a matrix composed of the corresponding canonical basis vectors $\mathbf{e}_k$ ($\forall k \in \mathrm{Q}$) arranged in rows. This optimization problem can be solved with the method of Lagrange multipliers. Specifically, we wish to find stationary points of the Lagrangian $L(\delta\mathbf{w}, \boldsymbol{\lambda})$, where $\boldsymbol{\lambda} \in \mathbb{R}^{|\mathrm{Q}|}$ denotes a vector of Lagrange multipliers. Solving the system of equations $\frac{\partial L(\delta\mathbf{w}, \boldsymbol{\lambda})}{\partial \delta\mathbf{w}} = \mathbf{0}$ and $\frac{\partial L(\delta\mathbf{w}, \boldsymbol{\lambda})}{\partial \boldsymbol{\lambda}} = \mathbf{0}$ yields the following optimal weight update:

$$\delta\mathbf{w}^* = -\widehat{\mathbf{F}}^{-1}(\mathbf{w}^*)\mathbf{E}_{\mathrm{Q}}^\top\Big(\mathbf{E}_{\mathrm{Q}}\widehat{\mathbf{F}}^{-1}(\mathbf{w}^*)\mathbf{E}_{\mathrm{Q}}^\top\Big)^{-1}\mathbf{E}_{\mathrm{Q}}\mathbf{w}^* \tag{3.5}$$

which prunes a set of weights Q and updates the remaining weights to preserve the loss. Now, the corresponding loss increase incurred by the optimal weight update $\delta\mathbf{w}^*$ can be expressed as the saliency score of weights Q, which we denote by:

$$\rho_{\mathrm{Q}} = \frac{1}{2}\left(\mathbf{E}_{\mathrm{Q}}\mathbf{w}^*\right)^\top \left(\mathbf{E}_{\mathrm{Q}}\widehat{\mathbf{F}}^{-1}(\mathbf{w}^*)\mathbf{E}_{\mathrm{Q}}^\top\right)^{-1} \mathbf{E}_{\mathrm{Q}}\mathbf{w}^*. \tag{3.6}$$

We use this saliency/importance score to rank groups of weights for pruning. As a sanity check, if we prune a single weight w_j at a time, our derivations will yield the standard formulas of [67], [68].

3.2.6 An Efficient Implementation

Implementing the previously described approach for LLMs, where the number of weights $\mathbf{w} \in \mathbb{R}^d$ is huge, is infeasible. In particular, this is due to the dependence on the inverse of the empirical Fisher information matrix $\widehat{\mathbf{F}}^{-1}(\mathbf{w}) \in \mathbb{R}^{d \times d}$, appearing in formulations of the saliency score and of the optimal weight update. We now describe how to circumvent these issues.

Pruning the optimal set of weights Assume a gradual pruning setup, in which at each pruning step, we wish to prune a model to a target sparsity $s \in (0, 1]$, effectively zeroing out $s \times d$ weights, in groups of size $|Q|$. Typically $s \times d \gg |Q|$, meaning we want to remove multiple groups simultaneously. Finding the optimal set of $\frac{s \times d}{|Q|}$ groups is an intractable combinatorial problem due to all possible correlations between them, given by the binomial coefficient $\binom{n}{k}$, where $n = \frac{d}{|Q|}$ and $k = \frac{s \times d}{|Q|}$. This problem can be alleviated by ignoring correlations between different groups of weights Q, and solving only for correlations between the weights within the same group. In practice, this boils down to evaluating the saliency score ρ_{Q} for each group Q, and pruning the $\frac{s \times d}{|Q|}$ groups with the lowest score. As pruning many weights in the same step can make the Taylor approximation of the loss function less accurate, one can consider pruning with multiple smaller sub-steps with re-computations of the Hessian approximation in between (without intermediate fine-tuning). While this can further improve the quality of the pruning step [120], we do not implement this additional optimization since the competing methods do not utilize re-computations.

Inverse empirical Fisher computation The above procedure's key space and time complexity cost is computing products with the inverse empirical Fisher. A direct approach would be to perform a block-wise diagonal approximation of this matrix (which we detail next), and perform direct block inversion. However, we found experimentally that this approach is too expensive in terms of time and quite numerically sensitive. As an alternative, we rely on the fact that the matrix we wish to invert is a sum of rank-1 matrices and employ the Woodbury/Sherman-Morrison (WSM) inversion formula. Specifically, given a sum $(\mathbf{A} + \mathbf{u}\mathbf{v}^\top)$ of an invertible matrix $\mathbf{A}$ and an outer product of vectors $\mathbf{u}$ and $\mathbf{v}$ with compatible dimensions, the inverse $(\mathbf{A} + \mathbf{u}\mathbf{v}^\top)^{-1}$ can be exactly calculated as $\mathbf{A}^{-1} - \frac{\mathbf{A}^{-1}\mathbf{u}\mathbf{v}^\top\mathbf{A}^{-1}}{1+\mathbf{v}^\top\mathbf{A}^{-1}\mathbf{u}}$. Placing the expression of the empirical Fisher in the WSM formula, we obtain the following recursive formulation, where m is the number of gradients employed in the approximation:

$$\widehat{\mathbf{F}}^{-1}(\mathbf{w}) = \widehat{\mathbf{F}}_m^{-1}(\mathbf{w}) = \left(\widehat{\mathbf{F}}_{m-1}(\mathbf{w}) + \frac{1}{m}\nabla\mathcal{L}_m(\mathbf{w})\nabla\mathcal{L}_m^\top(\mathbf{w})\right)^{-1} \tag{3.7}$$

Unrolling the recursion with $\widehat{\mathbf{F}}_0^{-1}(\mathbf{w}) = \frac{1}{\lambda}\mathbf{I}_d$, we can obtain an iterative formula to calculate the inverse of the empirical Fisher matrix as exactly

$$\widehat{\mathbf{F}}^{-1}(\mathbf{w}) = \widehat{\mathbf{F}}_m^{-1}(\mathbf{w}) = \frac{1}{\lambda}\mathbf{I}_d - \sum_{i=1}^{m} \frac{\left(\widehat{\mathbf{F}}_{i-1}^{-1}(\mathbf{w})\nabla\mathcal{L}_i(\mathbf{w})\right)\left(\widehat{\mathbf{F}}_{i-1}^{-1}(\mathbf{w})\nabla\mathcal{L}_i(\mathbf{w})\right)^\top}{m + \nabla\mathcal{L}_i^\top(\mathbf{w})\widehat{\mathbf{F}}_{i-1}^{-1}(\mathbf{w})\nabla\mathcal{L}_i(\mathbf{w})}. \tag{3.8}$$

The iterative formulation enjoys several computational advantages over the direct implementation. The most notable ones are 1) avoiding explicit calls to the expensive and dampening-sensitive matrix inversions and 2) allowing successive updates of the inverse as new gradients are computed, never needing to store all m gradients of size d and thus significantly reducing memory requirements.

3.2.7 Memory and Run-Time Complexity

Computing and storing the inverse empirical Fisher $\widehat{\mathbf{F}}^{-1}(\mathbf{w}) \in \mathbb{R}^{d\times d}$ is prohibitively expensive for modern LLMs, which have hundreds of millions of parameters, due to the quadratic complexity on the number of weights d. However, [68] have shown that a diagonal block-wise approximation of the empirical Fisher matrix can be very accurate for pruning convolutional neural networks. We adapt the same approach here, in the context of LLMs. Thus, for blocks of width B along the main diagonal, memory requirements for the computation of the inverse Fisher matrix are reduced from the quadratic $\mathcal{O}(d^2)$ to a linear $\mathcal{O}(Bd)$ dependence on the number of weights d. At the same time, run-time complexity relaxes from $\mathcal{O}(md^2)$ to $\mathcal{O}(mBd)$. As we will show, this computation can be efficiently and accurately performed for moderate values of m and B.

3.2.8 Efficient and Scalable Implementation

On the practical side, we have identified general hyper-parameters $B = 50$ for the block size and $m = 1024$ for the number of gradients that produce state-of-the-art results for all analyzed BERT models while still being able to fit on the 24GB RTX 3090 GPU. Moreover, for these parameter values, the block-wise approximation of $\widehat{\mathbf{F}}^{-1}(\mathbf{w})$ can be implemented very efficiently on modern accelerators. Specifically, we take advantage of the fact that such hardware favors batched matrix operations and that the blocks of size $B \times B$ in $\widehat{\mathbf{F}}^{-1}(\mathbf{w})$ are independent. With $N_B = \frac{d}{B}$, we refer to the total number of blocks, i.e. the batch-dimension. The procedure works as follows. First, we compute batched matrix-vector products $\widehat{\mathbf{F}}_{i-1}^{-1}(\mathbf{w})\nabla\mathcal{L}_i(\mathbf{w}) \in \mathbb{R}^{N_B \times B}$ and scalar denominators $m + \nabla\mathcal{L}_i^\top(\mathbf{w})\widehat{\mathbf{F}}_{i-1}^{-1}(\mathbf{w})\nabla\mathcal{L}_i(\mathbf{w}) \in \mathbb{R}^{N_B}$. Then, we update the inverse Fisher for each block by computing the scalar-scaled outer products. $\left(\widehat{\mathbf{F}}_{i-1}^{-1}(\mathbf{w})\nabla\mathcal{L}_i(\mathbf{w})\right)\left(\widehat{\mathbf{F}}_{i-1}^{-1}(\mathbf{w})\nabla\mathcal{L}_i(\mathbf{w})\right)^\top$ of shape $\mathbb{R}^{N_B \times B \times B}$.

3.2.9 Experimental Validation

To ease reproducibility, we conduct our experiments in modified versions of the popular open-source libraries: Transformers [123] and SparseML [124]. All of our experiments use publicly available datasets via [125] and focus on the BERT$_{\text{BASE}}$ model [9], one of the most commonly used LLMs, composed of 12 transformer layers with 110M parameters. Following community standards, we prune the encoder's weights (85M) and report sparsities relative to this number. Our models, compression recipes, and full implementation will be made public.

3.2.10 Downstream Unstructured Pruning

We first revisit the accuracy-compression trade-off for pruning on downstream tasks.

Goals and setup. We compare existing approaches, notably Movement Pruning (MvP) [66] and Lottery Ticket (LT-BERT) [24], against the gradual unstructured oBERT method, introduced in Section 3.2.4. Our experiments evaluate performance on a variety of downstream (English) tasks commonly used to evaluate model compression: question answering SQuAD v1.1 [7], sentence classification Quora Duplicate Query Dataset QQP [126], and natural language inference MNLI [127].

Comparison with MvP. For a fair comparison with MvP, we consider the 10-epoch gradual pruning setup used to obtain the best results by [66]. Specifically, we start from the $\text{BERT}_{\text{BASE}}$ model and perform 2 epochs of fine-tuning, 6 epochs of pruning, and 2 further epochs of fine-tuning of the compressed model. We impose a global sparsity distribution over all layers, prune with oBERT two times per epoch, and use KD from the fine-tuned $\text{BERT}_{\text{BASE}}$ teacher. For oBERT pruning, we use $m = 1024$ gradients, block size $B = 50$, and dampening $\lambda = 10^{-7}$ to approximate the inverse Hessian matrix. In all our runs, the first pruning step prunes 70% of weights and then follows the cubic interpolation [128] to the target sparsity. This sizeable first pruning step gives more time to recover from the later pruning steps, which impose higher sparsities.

We observe that Optimal BERT Surgeon outperforms Movement Pruning by a significant margin, more than 2 points of F1 score at the same sparsity. Remarkably, the model pruned with oBERT to 97% sparsity has similar accuracy to the MvP-pruned model at 90% sparsity, which has roughly 3x more weights. This reinforces the effectiveness of second-order information for pruning.

Extended pruning and fine-tuning. Next, we examine the effects of extending the gradual schedule to 30 epochs, matching the setup used for LT-BERT [24]. The only difference compared to our 10-epoch structure is that we now prune with oBERT every four epochs and rewind the learning rate after each pruning step. The extended setup leaves more time to recover from pruning. We report the mean over three runs. For additional evaluation metrics and standard deviations, please see Tables A.4 and A.7 in the Appendix. The results show a clear accuracy difference between oBERT and LT-BERT, especially at high sparsities. This difference is justified since the LT-based approach attempts to mainly transfer *network connectivity*, whereas the oBERT can also benefit from the weight values. Finally, we examined the impact of extended setup with Soft MvP on SQuAD, targeting 90% sparsity (not shown in the Table), leading to an (F1, EM) combination of $(87.42, 79.83)$ for MvP. The F1 gap in favor of oBERT is lower than at 10 epochs, suggesting that extended fine-tuning helps all methods, yet, it is far from negligible.

Task	BERT BASE	Spars.	Soft MvP	oBERT (ours)	LT-BERT	oBERT (ours)
	Epochs		10 Epochs		30 Epochs	
SQuAD F1	88.54	80%	-	-	86.54	**89.04**
		90%	84.90	**87.98**	68.00*	**88.31**
		97%	82.30	**84.65**	-	**85.98**
MNLI m-acc	84.54	80%	-	-	82.60	**84.32**
		90%	81.20	**83.20**	75.00*	**83.79**
		97%	79.50	**81.00**	-	**81.77**
QQP acc	91.06	80%	-	-	90.30	**91.57**
		90%	90.20	**90.89**	90.00	**91.35**
		97%	89.10	**90.23**	-	**90.87**

Table 3.1: Sparse-transfer dev-set performance of upstream-pruned $\text{BERT}_{\text{BASE}}$ models. (* approximate results as the exact numbers are not available.)

3.2.11 Upstream Unstructured Pruning

An appealing alternative to downstream pruning is to compress models upstream on the semi-supervised pre-training task [69]. Given the upstream pruned model, computational requirements for obtaining downstream fine-tuned models are significantly reduced, as only fine-tuning of the remaining weights is necessary.

Goals and setup. To compare with existing approaches, notably Prune OFA [69] and LT-BERT [24], we gradually prune with oBERT directly at upstream datasets, BookCorpus and English Wikipedia, and then fine-tune the remaining unpruned weights on the same three downstream tasks introduced in A.2.7.

Teacher preparation. Following [51], we start with the HuggingFace $\text{BERT}_{\text{BASE}}$ uncased model and fine-tune it for additional 10 epochs only on the masked language modeling task.

Pruning at upstream. Once the distillation teacher is trained, we gradually prune and fine-tune the $\text{BERT}_{\text{BASE}}$ model for 3 epochs, using KD from the dense teacher. We prune four times per epoch and rewind the learning rate to the initial value after each pruning step. Hyper-parameters for oBERT are the same as for downstream pruning in A.2.7.

Sparse-transfer to downstream. To evaluate the resulting upstream-pruned models, we finetune the unpruned weights on downstream tasks with KD from the fine-tuned $\text{BERT}_{\text{BASE}}$ model. For a fair comparison with Prune OFA, we fine-tune for eight epochs. The results in Table 3.2 show that sparse models produced by oBERT outperform state-of-the-art methods by significant margins. We report the mean over four runs. In contrast to Prune OFA, which performed extensive hyper-parameter tuning for sparse transfer, our

Task	BERT BASE	Sparsity	LT-BERT	Prune OFA	oBERT (ours)
SQuAD F1	88.54	90%	68.00*	87.25	**88.49**
		97%	-	-	84.92
MNLI m-acc	84.54	90%	75.00*	81.45	**83.40**
		97%	-	-	80.91
QQP acc	91.06	90%	90.00	90.93	**90.99**
		97%	-	-	90.33

Table 3.2: Sparse-transfer dev-set performance of upstream-pruned $\text{BERT}_{\text{BASE}}$ models. (* approximate results as the exact numbers are not available.)

recipe is simple and general across downstream tasks: 8 epochs of fine-tuning with a linearly decaying learning rate. This suggests that sparse pre-trained models found by oBERT constitute a strong starting point for sparse transfer learning, which can be further improved by task-specific hyperparameter tuning.

3.2.12 Compound Compression For CPUs

To probe the potential practical impact of our approach, we specialize the technique for deployment on CPUs, corresponding to "edge" deployments. Specifically, we tailor our sparse models to the DeepSparse [119] sparsity-aware runtime by compounding unstructured pruning with additional compression techniques.

Direct layer dropping. The competitive results obtained at high sparsities in sections A.2.7 and 3.2.11 suggest that $\text{BERT}_{\text{BASE}}$ may be overparameterized for downstream tasks. To improve the compression ratio and inference speed, we apply "direct" layer dropping: initially drop all but 3 or 6 of the BERT's 12 layers. We drop layers from our upstream teacher, following [129], and fine-tune them with KD in the same setup used to prepare the upstream teacher. These three and 6-layer models are used as starting points for downstream pruning. More sophisticated layer-dropping techniques [130] could bring further accuracy gains; we leave this for future work.

Block pruning and QAT. High-performance inference usually benefits more from (semi) structured sparsity patterns than unstructured ones. Hence, we employ the generalized oBERT formulation introduced in the section 3.2.4 and prune weights in the 4-block pattern, meaning that contiguous blocks of 4 weights are either set to zero or kept dense. Both pruning types, unstructured and 4-block, can be leveraged for computational speedups with the DeepSparse runtime, but 4-block pruning coupled with INT8 quantization can pro-

Layers	Sparsity	Unstructured	4-block	+QAT
12	0%	89.48	89.48	89.06
	80%	89.04	88.57	87.89
	90%	88.31	87.57	86.68
6	0%	88.32	88.32	87.94
	80%	88.20	87.00	86.10
	90%	86.78	85.34	84.59
3	0%	84.66	84.66	84.25
	80%	84.08	82.79	82.04
	90%	82.50	80.69	79.66

Table 3.3: F1 score of the 3, 6, and 12-layer models compound-compressed on the SQuADv1.1.

vide further performance gains. For quantization, we apply standard quantization-aware training (QAT) [131] on top of the 4-block models.

Compounding for deployment. To determine the impact of different compression schemes, we investigate unstructured and 4-block pruning of the 3, 6, and 12-layer models. We use the same hyper-parameter set for all runs from the extended pruning and fine-tuning setup in Section A.2.7. The results are given in Table 3.3, where we also report the accuracy of the corresponding dense models (0% sparsity) in the same setup. The results indicate that compression methods can be combined without model collapse, although the accuracy drops do compound. The fact that highly compressible layer-dropped models suggest that structured and fine-grained (unstructured) compression are complementary. We find it remarkable that our 6-layer unstructured oBERT-pruned model is competitive with the 12-layer MvP-pruned model when both are pruned to 90% sparsity.

Practical trade-offs. We now benchmark these models in an end-to-end fashion, both in terms of model size and inference speed. For model size, we report the checkpoint size in MB after standard gzip compression. We say the number of items per second (throughput) for inference speed on the well-established SQuAD v1.1 CPU-inference benchmark with a sequence length of 128 and a batch size of 32. Figure 3.2 depicts relative accuracy versus magnitude of improvement in speed and model size. As a baseline for full recovery, we follow the community standard, e.g. [66], and adopt the dense BERT$_{\text{BASE}}$ model with an 88.54 F1 score. The baseline for inference speed is dense BERT$_{\text{BASE}}$ inference with DeepSparse, which matches the industry-standard ONNX Runtime inference engine. Results suggest a roughly-linear trade-off between compression and accuracy loss, with a compression jump of around 1% accuracy drop due to the application of quantization. Specifically, we observe 8.4x higher

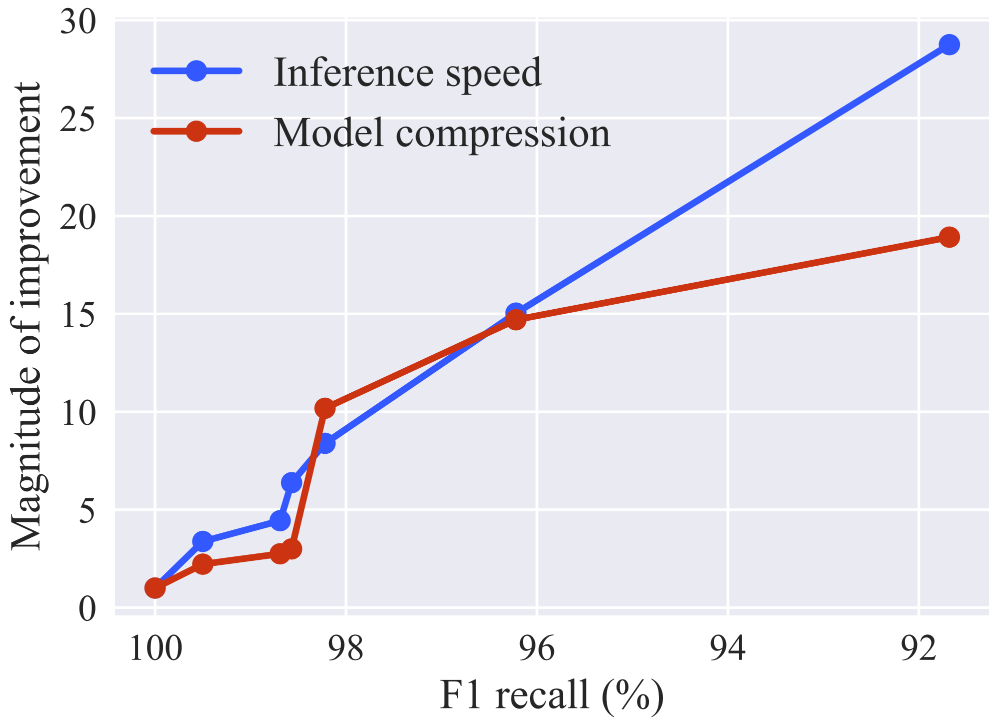

Figure 3.2: F1 recall on the SQuADv1.1 task relative to improvements in CPU-inference speed and model size.

inference speedup at less than 1% accuracy drop, 10x speedup at less than 2% drop, 15x speedup at less than3% depth, and 29x speedup at less than 7.5% accuracy drop. This shows how compound compression can optimize LLMs to various latencies.

3.2.13 Discussion

Broader comparison. We now contrast our compound-compressed BERT$_{\text{BASE}}$ models to alternative compression techniques, which achieve practical speedups. We compare against DistilBERT [55], TinyBERT [77], and Block Pruning For Faster Transformers (Hybrid Filled MvP) [75]. DistilBERT leverages KD from the 12-layer model during pre-training and fine-tuning to obtain a 6-layer model fine-tuned for a specific downstream task. We employ a similar approach for our 6-layer model but apply KD only for the teacher's output and the oBERT pruning to improve accuracy-compression trade-offs further. TinyBERT uses a specialized Transformer-layer KD scheme to distill knowledge and intermediate representations from the 12-layer teacher at both stages, pre-training and fine-tuning on a specific task. In contrast, we use a more straightforward approach and only employ KD from the teacher's outputs. Hybrid Filled MvP, the best-performing method in [75], employs semi-structured pruning and weight reintroduction. The comparison is given in Table 3.4, where we present the number of unpruned encoder weights as size, compression ratio relative to the dense $\text{BERT}_{\text{BASE}}$, inference speedup relative to the dense $\text{BERT}_{\text{BASE}}$ evaluated in the same inference environment, and F1 score on the dev-set of the SQuAD v1.1 dataset. The results suggest that our compressed models improve upon the current state-of-the-art techniques, setting new, very competitive baselines concerning all metrics: accuracy, model size, and inference speed.

Model	Size	Compr.	Speedup	F1	Dev.
$\text{BERT}_{\text{BASE}}$	85.0M	1.00x	1.00x	88.54	
¡ 6-layers					
TinyBERT_4	4.5M	18.88x	9.40x	82.10	GPU
$\text{oBERT}_{3,90}$	**2.1M**	**40.00x**	**14.80x**	**82.50**	CPU
6-layers					
DistilBERT	42.5M	2.00x	2.00x	86.90	GPU
TinyBERT_6	42.5M	2.00x	2.00x	87.50	GPU
$\text{oBERT}_{6,80}$	**8.5M**	**10.00x**	**6.38x**	**88.20**	CPU
12-layers					
Hybrid F. MvP	30.7M	2.76x	1.84x	88.70	GPU
$\text{oBERT}_{12,80}$	**17.0M**	**5.00x**	**3.38x**	**89.04**	CPU

Table 3.4: Compressed $\text{BERT}_{\text{BASE}}$ models on the SQuADv1.1 task. ($\text{oBERT}_{6,80}$ stands for the 6-layer model pruned to 80% sparsity.)

In terms of future work, we aim to investigate the distillation of intermediate model representations [79] and the role of compound compression applied to much larger language models, including generative ones.

3.2.14 Broader Impact

Our work is part of the general trend of producing efficient inference models that approximate the performance of their larger bases. This work should help increase model efficiency, thereby reducing the computational and, ultimately, the monetary cost of executing such models. Moreover, it could allow models to be used by those who do not have access to expensive specialized computing clusters. For instance, our main speedup results are aimed at widely-available CPUs.

3.2.15 Limitations

As with any academic study, our work is not without its limitations. We split their discussion into limitations that are *inherent to our method* and limitations *of our present* *study*; extensions of our work can overcome the latter. In the first category, we highlight that our second-order method relies on approximations inherent to scaling such practices to the BERT scale. Prior studies, e.g. [68], have carefully examined the validity of these approximations in the context of Convolutional Neural Network (CNN) models. The strength of our empirical results can be seen as indirect evidence that these approximations also apply to BERT models. A second technical limitation is that our method requires non-trivial additional storage costs; At the same time, we have shown that our experiments can be executed on a single commodity GPU (NVIDIA RTX 3090); this limits the range of devices to which the technique may be applied. However, we provide an efficient and easy way to scale our approach with more GPUs, automatically utilized in a multi-GPU environment.

Regarding the limitations of our present study, we note that we have only considered compression for the classic $\text{BERT}_{\text{BASE}}$ model. Our preliminary experiments, not included due to space limitations, suggest that the same technique can extend to other models, such as DistilBERT; we have focused on $\text{BERT}_{\text{BASE}}$ because it has emerged as a consistent benchmark for unstructured pruning methods, and it allows us to compare plans in a consistent setting. Another limitation we aim to remove in future work is the focus on relatively fine-grained sparsity types, such as unstructured and semi-structured pruning.

3.3 SPARSE*BERT: SPARSE MODELS GENERALIZE TO NEW TASKS AND DOMAINS

3.3.1 Overview

Large Language Models have become the core architecture upon which most modern natural language processing (NLP) systems build. These models can consistently deliver impressive accuracy and robustness across tasks and domains, but their high computational overhead can make inference difficult and expensive. To make using these models less costly recent work has explored leveraging *unstructured pruning* to improve inference speed and decrease size. This section studies how models pruned using Gradual Unstructured Magnitude Pruning can transfer between domains and tasks. Our experimentation shows that pruned models using general domain masked language models during pretraining can transfer to new domains and tasks without extensive hyperparameter explorations. We demonstrate that our general sparse model *Sparse*BERT* can specialize simply by pretraining the compressed architecture on unstructured biomedical text to become SparseBioBERT. SparseBioBERT can match and exceed the performance of BioBERT with 10% of the parameters.

3.3.2 Introduction

Foundational Models [132] based on the Transformer architecture [36] has quickly become the most common building block in the modern language understanding stack, providing robust language representations which can be leveraged to provide impressive accuracy on tasks like question answering, text classification, and token classification. These Large Language Models (LLMs) can adapt to novel domains through pretraining resulting in models like BioBERT [133], LegalBERT [134], and SciBERT [135] has become a popular strategy for improving performance further. While accurate and robust, LLMs are not without drawbacks. They commonly have hundreds of millions or billions of parameters requiring large specialized computer clusters to run inference at scale. Several approaches have been successfully used to improve the performance of these LLMs, such as approximating attention [136], removing portions of the models [72], and reducing the precision of activation and weight values.

Recent work [137] [138] has shown that the application of unstructured and semi-structured (block) pruning mechanisms on LLMs can significantly compress models with little to no loss in accuracy. While these approaches are successful and applicable during model general domain pretraining and task-specific fine-tuning, prior work has not studied how pruned models transfer to new domains nor the impact of pretraining stage pruning on transfer accuracy. Given that most applications would require the transfer of the general domain LLMs to a specific application domain, it is important to study the generality and robustness of the pruned LLMs when applied to multiple tasks in an application domain.

While existing pruning research has found it possible to prune models heavily without loss in accuracy, most approaches have focused on the compression of individual tasks or textual domains. These specialized models match or exceed the accuracy of the dense model but commonly require vast amounts of hyperparameter turning and task-specific optimization to achieve this result. Compressed models like DistillBERT [55] and TinyBERT [77] are some of the most popular LLMs because they provide compression without any additional know-how or optimization. For pruned models to become a common architecture for NLP tasks, they must be as robust and easy to use as their uncompressed counterparts.

This section explores this potential and proposes Sparse*BERT, a new pruned LLM that can adapt effectively to new domains without extensive fine-tuning or task-specific pruning. Our work studies generalizable pruned LLMs by evaluating how well they can transfer to previously unstudied tasks in the biomedical domain. Specifically, we study these questions by focusing on transferring pruned and unpruned LLMs to the biomedical domain and evaluating the accuracy of said models on downstream tasks like Entity Extraction(EE), Relation

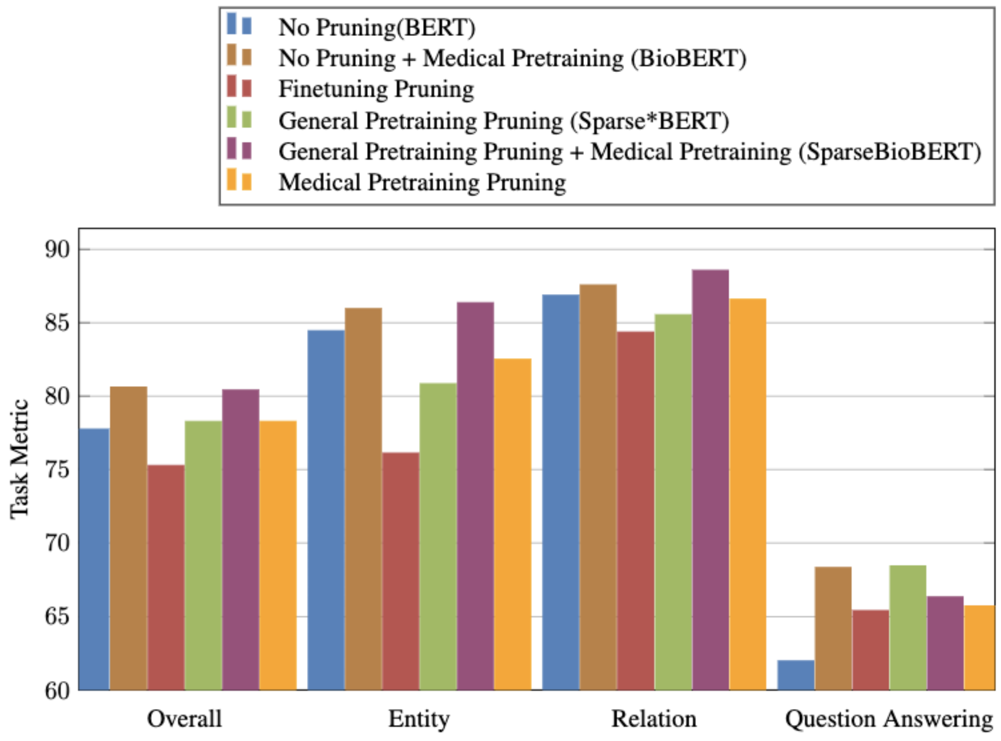

Figure 3.3: Impact of stage and domain of pruning when evaluating BERT base uncased on medical NLP Entity Extraction, Relation Extraction, and Question Answering. Medical pretraining can improve performance for pruned and unpruned models, and the performance of the pruned model and the performance of BioBERT and SparseBioBERT is comparable.

Extraction(RE), and Question Answering(QA). Our experiments demonstrate that pruned LLMs generalize well and are robust to domain transfer and variation in target task, dataset size, and dataset difficulty. In summary, our contributions are as follows:

- We reinforce Zafrir et al.'s findings that LLMs pruned on the general domain data can transfer to new domains without extensive hyperparameter tuning and extend their work, demonstrating these pruned models can be transferred to new pre-training domains without additional parameter optimization.

- We introduce a sparse model, Sparse*BERT, and its domain adaptation adapted for the Medical/Bio NLP domain called SparseBioBERT. This model matches the accuracy of the BioBERT with 10% of the active parameters.

3.3.3 Background and Related Work

Large Language Models such as language representation and generation models commonly use multiple layers of transformer encoders or decoders. Each transformer layer usually contains some form of multi-head attention (MHA) and fully-connected feed-forward networks (FFN). The MHA is made up of multiple self-attention heads [36], each of which has 3 sub-components: queries (**Q**), keys (**K**), and values (**V**). Equation 3.9 shows the expression used to compute the *attention* of each *head*, where d is the dimensionality of **K**. The output of the attention heads is concatenated and fed into the FFN.

$$Attention(Q, K, V) = softmax\left(\frac{QK}{\sqrt{d}}\right)V \tag{3.9}$$

Attention, while simple, has proven to be incredibly robust as it allows models to scale to hundreds of layers, hundreds of attention heads [2], and seemingly most modalities [139] [140]. Despite its generalization ability, attention-based models are also brittle as removing less than 0.0001% of parameters can cause complete model collapse [141].

Unstructured pruning compresses a model by removing individual weights from a network by setting them to zero. Pruning methods commonly remove weights based on their saliency to the network and, to avoid model collapse, usually do so gradually while fine-tuning the remainder of the weights. Since it is difficult to quantify the true saliency of weight concerning a network, zeroth, first, and second-order estimation methods exist to approximate saliency.

Zero-order methods use weight magnitudes as a proxy, *i.e.*., remove the smallest weights without evaluating the impact of their removal on model accuracy. These approaches are prevalent for Convolution Neural Networks [57] and have recently been successful for LLM [142] [24] [137]. First-order methods like Movement Pruning [66] use a gradient-based approximation to remove the weights moving toward zero. Second-order methods like OBS [68] estimate the impact of individual weight removal via approximations of second-order derivatives and use it as a proxy for saliency.

Using unstructured and semi-structured pruning has proved to be a convenient way of compressing LLM for efficient inference and decreased model size. For example, a BERT-Base-Uncased model which has had 90% of its parameters pruned runs inference ∼4.5 times faster and is 2.75 times smaller with no drop in accuracy [138]. If this compression leverages additional methods like quantization and layer dropping, inference speed can improve by 28.75 times, and model size can drop by ∼19 times.

While many successful compression approaches exist, Transformer models are fragile [60], as minor perturbations can lead to model collapse. Moreover, there is a lack of understanding of the generality and robustness of compressed LLMs when transferring them to different tasks in an application domain, a gap our work attempts to fill.

3.3.4 Sparse*BERT: General Sparse Models Can Adapt to New Domains

We can formulate the Sparse*BERT model as θ^*, which can approximate the accuracy of the dense model θ and does not suffer from model collapse when transferred to new domains. The architecture of Sparse*BERT matches BERT, but a portion of its weights are pruned and masked to avoid future updates. To ensure that our sparse model can approximate the accuracy of the dense model, we leverage Knowledge Distillation between the outputs of the dense and sparse networks.
Following the success of Zafrir et al. [137], we leverage Gradual Magnitude Pruning (GMP) as shown in algorithm 3.1. Pruning models using gradual magnitude pruning are quite effective and easy to use. At its core, the goal of GMP is to remove weights slowly enough that the network can learn to recover after each pruning step.
To ensure that we can maximize the inference speedups, we prune each set of components in the network independently. The structure in the model graph groups these components, so the individual feed-forward layers, the queries, or keys, but not individual self-attention heads.

Algorithm 3.1: Uniform Gradual Magnitude Pruning

```
$\theta_0$ a pretrained neural network, $\theta_t$ a dense pretrained neural network (distillation teacher),
  $D$ a training dataset, $N$ number of pruning steps, $\sigma$ weights to prune at each pruning
  step a pruned neural network **for** *x in 1 to N* **do**
**end**
$\theta^* \leftarrow \theta_0$ **for** *component in* $\theta*$ **do**
**end**
$w \leftarrow sort(component)$ $\theta^* \leftarrow prune(\theta^*, w, \sigma)$ $\theta^* \leftarrow train(\theta^*, \theta_t, D)$
return $\theta^*$
```

3.3.5 Experiments

To assess how well-pruned models can transfer to new domains and determine the optimal stage of pruning (pretraining, domain transfer, or fine-tuning), we evaluate the accuracy of 10 Biomedical biomedical datasets. Our experiments fix the training parameters for all the transfer tasks and vary the stage used for pruning (no pruning, pretraining, domain transfer, fine-tuning) and domain-specific pretraining.

3.3.6 Datasets

In all of our experiments, we focus on English-only biomedical datasets/corpora. Each dataset we use is unmodified from its described use in the literature, and its associated and prescribed metrics are used.

3.3.7 Finetuning Datasets

Dataset	Domain	Task Type	Training Size	Testing Size	Validation Size	Evaluation Metric
BC5-Chem	Medical	Entity Recognition	5203	5347	5385	F1
BC5-disease	Medical	Entity Recognition.	4182	4244	4424	F1
NCBI-disease	Medical	Entity Recognition	5134	787	960	F1
BC2GM	Medical	Entity Recognition	15197	3061	6325	F1
JNLPBA	Medical	Entity Recognition	46750	4551	8662	F1
ChemProt	Medical	Relation	18035	11268	15745	F1
DDI	Medical	Relation	25296	2496	5716	F1
GAD	Medical	Relation	4261	535	534	F1
PubMedQA	Medical	Question Answering	450	50	500	Accuracy
BioASQ	Medical	Question Answering	670	75	140	Accuracy
SQUAD	General	Question Answering	87599	10570	N/A	F1

Table 3.5: To understand how generalizable sparse models are, we evaluate a wide set of tasks that vary in difficulty, size, and desired output

Pretraining Datasets. To understand how the stage of pruning impacts model accuracy, we train models both pruned and dense models on the Medline/PubMed corpus and the combination of English Wikipedia [143] and The Book Corpus [144] datasets.
The combination of Wikipedia and Book Corpus datasets creates a common domain language dataset featuring 3.3 billion words, which have become the backbone for general domain masked language modeling experimentation.
The MEDLINE/PubMed Corpus is a publicly available[11] text corpus made up of journal abstracts and documents of biomedical literature from around the world. The United States National Institute of Health updates the corpus daily and has been the primary resource used to train Biomedical LLMs like BioBERT [133] and PubMedBERT [145]. For our experiments, we extracted our corpus on January 2022, filtered and prepared the dataset for masked language modeling using the BioElectra's [146] scripts[12]. This formatted PubMed

[11]https://www.nlm.nih.gov/databases/download/pubmed_medline.html
[12]https://github.com/kamalkraj/BioNLP-Corpus

corpus has 34,246,348 abstracts and 4.5 billion words [146].

Finetuning Datasets. We finetune pretrained models on 10 established Biomedical NLP datasets, encompassing 3 separate task types: Entity Recognition (ER), Relation Extraction (RE), and Question answering (QA). For ER, we use the BioCreative II Gene Mention Recognition (BC2GM), [147], BC5CDR Drug/Chemical (BC5-Chem), BC5CDR Disease (BC5-Disease) [148], JNLPBA [149], and NCBI Disease [150] datasets. For RE we use ChemProt [151], Drug-Disease Interaction (DDI) [152], and Gene-Disease Associations (GAD) [153] datasets. We leverage BioASQ task 7B [154] and PubMedQA [155] for QA. In addition, we analyze the impact of the finetuning dataset's size on the optimal pruning stage using the non-biomedical QA SQUAD [7] dataset. Details on the dataset size, evaluation metric, and domain can be found in Table 3.5.

3.3.8 Models and Experimental Setup

Our experiments focus on the popular BERT-base-uncased language model [9], an LLM composed of 12 transformer encoder layers with 110M parameters. We compare the performance of our sparse language models to the dense, task-tuned BioBERT model, which is also a 12 transformer layer model which features cased (BioBERT-Base-Cased) and uncased (BioBERT-Base-Uncased)variants. Following previous work, we do not prune the embedding layers of the network or any task-specific heads and focus on the ~ 85 million parameters found in the encoder layers. To ensure that our experiments are reproducible [13] we use the popular open-source libraries SparseML [14] and Transformers [15].

Model Pretraining Pretraining is when the model is trained on an unsupervised NLP dataset using a masked language modeling (MLM) approach [9]. Pretrained models are fine-tuned on labeled task-specific datasets to optimize for task-specific accuracy.

Our experiments use existing dense pretrained models for BERT-base-uncased [9] and PubMedBERT [145] and prune them using gradual magnitude pruning based on the corresponding dataset and MLM approach.

During model pretraining, we train for three epochs on 4 A100 GPUS using a batch size of 256 and a sequence length of 512, and, following early experiments and findings from Kurtic et al. and Zafir et al., we cycle the learning rate during pretraining and found cycling twice per epoch from 5e-4 to 0 to be most effective. First, we take the existing dense

[13]we will be releasing our code and models to allow reproducibility and extensibility

[14]https://github.com/neuralmagic/sparseml

[15]https://github.com/huggingface/transformers

models and run this training setup for 3 epochs to ensure model convergence; then, taking these converged models, we retrain and apply gradual magnitude pruning over the first two epochs. During pruning, we start from an initial sparsity of 30% and gradually prune to a final sparsity of 90%, pruning 100 times an epoch. After model pruning, we continue to train for one additional epoch to ensure that the sparse model is converged. Based on early experiments, we find knowledge distillation beneficial. For all of our experiments in pre-training, we leverage well-trained dense teachers using a hardness of 0.5 and a temperature of 2. When pruning weights, their values are fixed to 0 and are masked to avoid future updates. This means that our experiments effectively evaluate the discovery of the most optimal sub-architecture.

Model Finetuning the stage of model optimization using a supervised dataset using a task-specific training regime. In this stage, one or many classification heads have connected the model, and these classification heads and the pretrained model are trained for optimal performance on an individual task.

We fix the training procedure across fine-tuning tasks to isolate the effects of task-specific hyperparameter tuning and pruning stages. Specifically, we train each model for ten epochs on a V100 GPU using a batch size of 16, a learning rate that linearly decays from 5e-5 and replicates using five random seeds for larger tasks and ten random seeds for smaller tasks.

We use the same setup for fine-tuning already pruned models and when applying grad-

Model	Pruning Stage	EE	RE	QA	Overall
BERT-Base-Uncased	None	**84.44**	**86.86**	61.97	77.76
	Fine-tuning	76.12	85.36	65.39	75.28
	Pretraining	80.84	85.54	**68.44**	**78.27**
BioBERT-Base-Uncased	None	85.96	87.56	**68.33**	**80.62**
	Fine-tuning	63.53	75.14	54.00	66.34
	Pretraining (Medical)	82.50	86.60	65.71	78.27
	Pretraining (General)	**86.36**	**88.57**	66.33	80.42

Table 3.6: Overall results on the impact of task and dataset of model pruning. Models trained for the general domain and pruned on the general domain can transfer at equal or better accuracy. Question Answering is the notable outlier as its small dataset size benefits from the sparse models as their pruned architecture prevents overfitting on small datasets.

ual magnitude pruning during fine-tuning (pruning on the fine-tuning stage). For pruned models, we preserve the sparsity patterns. When pruning models during fine-tuning, we fine-tune the dense model for two epochs, prune over the preceding six epochs, and stabilize the pruned network for two epochs. In our early experiments and matching prior findings [137], we find that when pruning models on transfer tasks, accuracy is best when the learning rate cycles. Cycling only occurs when pruning during fine-tuning, and the learning rate cycles at epochs 2 (start of pruning) and 8 (end of pruning). Unlike previous work, we do not find a significant effect in accuracy improvement by leveraging knowledge distillation on the fine-tuning task. As a result, we do not use knowledge distillation during fine-tuning.

3.3.9 Experimental Results

Model	Pruning Stage	BC5-Disease	BC5-chem	NCBI-disease	BC2GM	JNLPBA	ChemProt	DDI	GAD	PubMedQA Accuracy	BioASQ Accuracy
Training dataset size	N/A	4182	5203	5134	15197	46750	18035	25296	4261	450	670
BERT-Base-Uncased	None	80.60	91.23	**85.66**	81.97	81.56	88.19	94.35	78.05	47.46	76.65
	Fine tuning	69.87	81.72	75.57	74.27	79.57	85.41	92.72	74.88	52.67	78.11
	Pretraining	75.35	87.83	81.75	77.12	81.24	86.13	92.73	77.77	50.00	82.89
BioBERT-Base-Uncased	None	83.195	93.63	83.46	86.05	**84.10**	90.66	95.01	77.02	**54.00**	**82.67**
	Fine tuning	66.34	66.60	52.69	59.71	62.15	80.28	87.60	57.55	**54.00**	**82.67**
	Pretraining(General)	**83.32**	**93.81**	84.15	**87.04**	81.84	**90.71**	**95.02**	**79.90**	51.50	81.17
	Pretraining(Medical)	80.56	92.17	78.84	81.11	81.81	88.02	93.99	77.77	49.14	82.28

Table 3.7: Performance on Complete set of tasks. Except for question-answering tasks and NCBI-Disease, the SparseBioBERT outperforms all other models, including BioBERT, indicating that sparse architectures can be transferred to new domains and use cases without additional optimization.

When we evaluate results on an average of task-specific scores as shown in Table 3.6, we can see that the SparseBioBERT model performs on par with its unpruned counter-part and outperforms it on relation extraction and entity extraction. Results on individual tasks can be found in table 3.7 and further consistently show how SparseBioBERT approximated BioBERT. When pruned models do not transfer to the biomedical domain, they can perform much worse than the unpruned models, as shown by the sizable gap between the pruned and dense BERT-base-uncased models. This result, coupled with the performance of Sparse-BioBERT, makes us believe that pruned models can adapt to new domains like unpruned models but require additional domain-specific pretraining to ensure performance.

We believe these results prove that models pruned during general domain data can remove large portions of the model without affecting the ability to transfer to new domains or tasks. Unlike pruning on specific data domains and tasks, general domain data pruning can preserve accuracy invariant to task and domain.

Unexpectedly, when evaluating biomedical QA, we improve accuracy by pruning, but only on a general domain language model pruned downstream or the general domain Sparse*BERT. We attribute this to the regularizing effect that pruning can have, and it likely helps in overfitting small datasets, PubMedQA and BioASQ. Tasks. Finding those models pruned outperform all others, and we believe the regularization provided by pruning can prevent overfitting on these small datasets.

Our results also indicate that it is optimal to prune with general domain data and transfer it to new tasks and domains for optimal performance. Regardless of their domain expertise,

BERT and BioBERT both see huge losses in accuracy when pruned on the downstream tasks, and these same losses are not found in the model pruned during pretraining. Surprisingly, the model pruned on the general domain data outperforms when pruned on biomedical domain-specific language modeling. This gap is nearly 4 points on entity extraction and 2 points overall, almost more significant than the gap between the BERT and BioBERT.
When we evaluate the impact of pruning on individual tasks pruning in fine-tuning stage) as shown in Table 3.7, we can see that pruning is quite sensitive to the dataset task. Looking at the large datasets like JNLPBA in Table 3.7, there is nearly no distinction in pruning during pretraining or fine-tuning. On the other hand, small datasets like NCBI and GAD see a large accuracy loss from models pruned during fine-tuning.

3.3.10 Impact of Training Data Size

Noting that there is a significant variation in dataset size in the biomedical NLP tasks, we leveraged a dataset well studied in pruning, SQUAD [7], and performed variations to the training data size. Starting with the initial training dataset size, 88,000 items, we decreased the size to 75%,50%,25%,10%,5%,2%,1% and evaluated the impact to performance. We compared the dense BERT, Sparse*BERT, and pruning BERT during fine-tuning. The sparse models each have 90% unstructured sparsity on the encoder portion of the LLM.
Each experiment was performed with five random seeds, using a batch size of 12, and trained for 30 epochs with a learning rate decaying linearly from 5e-5 to 0. Each model's training uses knowledge distillation from a dense teacher with the hardness set to 0.8 and the temperature to 2.0. For the model pruned during fine-tuning, we cycle the learning rate at the beginning of pruning(2nd epoch) and the end of pruning(20th epoch). We evaluate model performance on the F1 score on the unaltered dev portion of the SQUAD dataset to avoid losses in evaluation sensitivity.
Figure 3.4 and table 3.8 demonstrate that models pruned during finetuning are not robust to variations in data size. Model performance decays slowly from 85 to 80 until the training data is decreased by 75%, but it quickly becomes nearly unusable when it becomes smaller than that. The same cannot be said about the dense or the Sparse*BERT model, as they see virtually identical losses in quality from

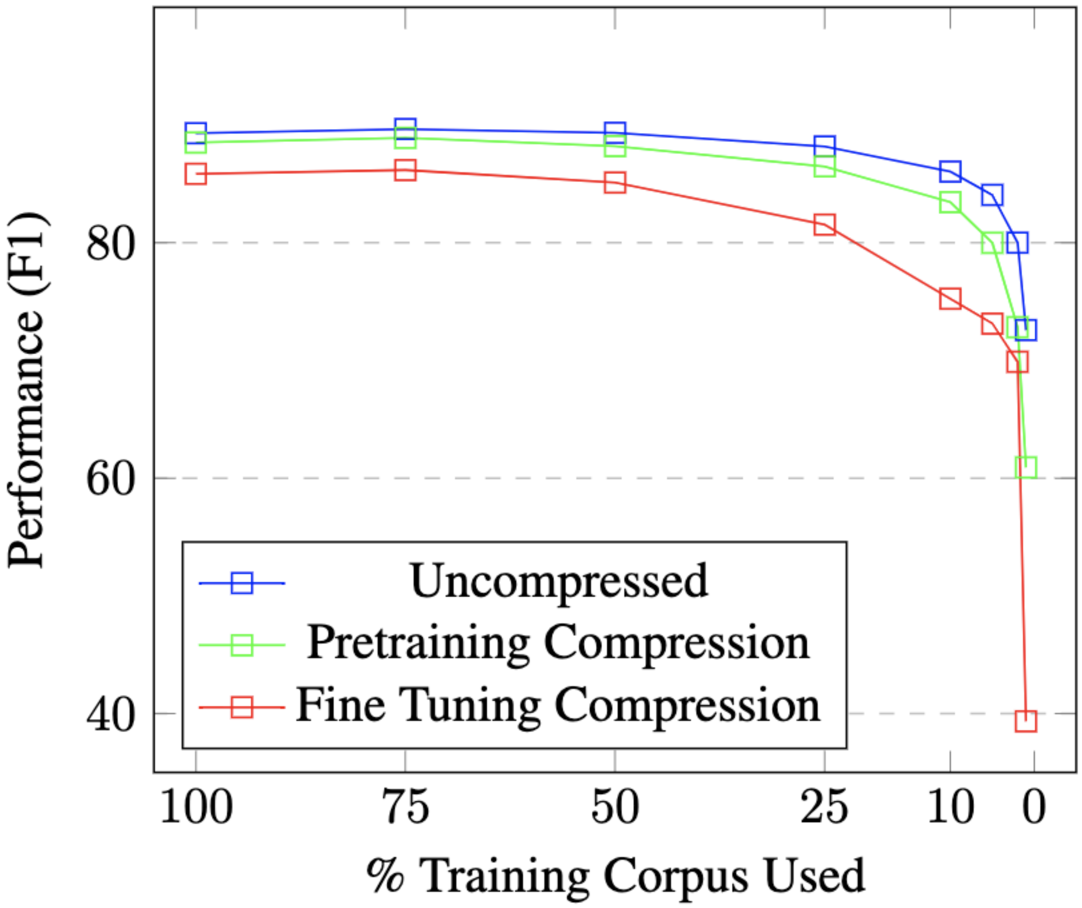

Figure 3.4: Impact of varying the training data size with pruned and dense models showcases how pretraining pruning has similar accuracy to the dense models.

Training Data Portion	BERT-BASE-Uncased	Sparse*BERT	Finetune Pruning
100	89.30	88.50	85.852
75	89.65	88.91	86.17
50	89.33	88.21	85.11
25	88.17	86.48	81.5
10	86.04	83.45	75.24
5	84.06	79.99	73.13
2	80.00	72.83	69.88
1	72.58	60.91	39.35

Table 3.8: Model accuracy as measured by F1 on dev portion of SQUAD compared to model type and the impact of training data size. Sparse models are not as sample efficient as their dense counterparts, but Sparse*BERT performance matches the dense model much more than the model pruned downstream.

3.3.11 Limitations

Our approach is limited in the computational time required to generate a general sparse LLM and the diversity of types of LLMs that we explore.
Regarding computational expense, training a sparse model requires negligible additional resources, which is tractable for models with a hundred million parameters and a few billion tokens, not for billion parameter models commonly discussed.
Our current explorations have been limited to monolingual LLMs trained in English. It is unclear how well sparse architectures will perform in a multilingual setting, and we expect degradation in language quality to be anything but equal across all languages.

3.3.12 Conclusion

In this section, we have introduced Sparse*BERT, a pruned LLM which builds on successful pruning algorithms and demonstrates its ability to transfer to new domains and tasks without additional hyperparameter search. Our experiment illustrates how well Sparse*BERT can move to the biomedical field to become SparseBioBERT. SparseBioBert can match the performance of BioBERT with $\frac{1}{10}$ of the parameters and no additional task-specific optimization. In future work, we seek to expand Sparse*BERT to new legal, financial, and medical domains. Furthermore, we like to continue our work on more complex models studying how sparsity impacts multilingual and multi-task models. In particular, we seek to understand how structured and unstructured approaches in compression relate to the curse of multilingualism.

3.4 OBERTA: IMPROVING SPARSE TRANSFER LEARNING VIA IMPROVED INITIALIZATION, DISTILLATION, AND PRUNING REGIMES

3.4.1 Overview

In this section, we introduce the range of oBERTa language models, an easy-to-use set of language models which allows Natural Language Processing (NLP) practitioners to obtain between 3.8 and 24.3 times faster models without expertise in model compression. Specifically, oBERTa extends existing work on pruning, knowledge distillation, and quantization and leverages frozen embeddings, improves distillation, and model initialization to deliver higher accuracy on a broad range of transfer tasks. In generating oBERTa, we explore how the highly optimized RoBERTa differs from the BERT for pruning during pre-training and fine-tuning. We find it less amenable to compression during fine-tuning. We explore the use of oBERTa on seven representative NLP tasks and find that the improved compression techniques allow a pruned oBERTa model to match the performance of BERT$_{\text{base}}$ and exceed the performance of Prune OFA Large on the SQUAD V1.1 Question Answering dataset, despite being 8x and 2x respectively faster in inference.

3.4.2 Introduction

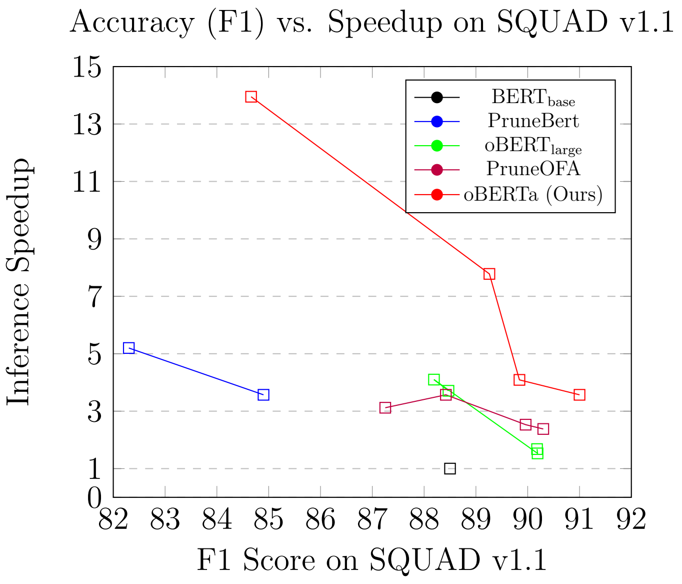

Figure 3.5: Performance of Sparse Language Models on the SQUAD V1.1 [156] compared to an uncompressed BERT$_{\text{base}}$ [9] with relation to realized inference improvements with regards to mean latency with a batch size of 1.

The massive improvement in contextual word representations driven by the usage of the Transformer architecture [36] has led to the wide-scale deployment of language models. These models are customized for various use cases and tasks like question answering, sentiment analysis, information retrieval, and document classification and deployed into general domains and specialized domains such as financial, medical, and legal. While these models are effective, they commonly contain hundreds of millions of parameters, which can lead to slow inference times without using specialized hardware accelerations like graphics processing units (GPU) or Tensor Processing Units (TPU). Without hardware acceleration, the inference on CPUs can be slow and impractical for real-world deployments.
Approaches such as knowledge distillation (KD) [76], quantization [81], and pruning [25] have been leveraged to improve model efficiency and, when paired with specialized inference engines[16], it is possible to accelerate inference times on CPUs and GPUs significantly. While there has been substantial effort to create effective methods for compression [77], [78] and improved model performance [51], general users of language models have been slower to adopt these methods. Years after its release, the original BERT$_{\text{base}}$ uncased [9] is still the most popular language model [17], followed by the slightly compressed DistilBERT [55] for latency-sensitive deployments. To enable broad adoption, regular users must be able to leverage more efficient language models without additional compression steps or tuning.
We present a case study on how to compress a language model for efficient CPU inference leveraging KD, structured pruning, unstructured sparsity, and quantization such that the compressed models can be applied to a broad range of natural language processing (NLP) tasks without expertise in compression of language models.
As part of this study, we release a set of efficient language models optimized to deliver the greatest improvement in inference while minimizing losses in accuracy. We then show how these models can be used for *sparse transfer learning* [137], [157] such that most compression happens during the pre-training stage. The pre-trained sparse models can be transferred to various NLP tasks, preserving sparsity without extensive optimization. Using these sparse transfer models and the DeepSparse inference engine, we show these sparse models can be fine-tuned to produce task-specific sparse models with minimal accuracy loss and result in greatly improved inference speeds with minimal accuracy loss.
As shown in figure 3.5, oBERTa provides state-of-the-art performance for sparse language models on the SQUAD v1.1 Question Answering dataset. oBERTa variants exceed the performance of BERT$_{\text{base}}$ despite being eight times faster, exceed the performance of Prune OFA$_{\text{large}}$ and oBERT$_{\text{large}}$ while being two to five times faster. In this section, we focus on

[16]https://github.com/neuralmagic/deepsparse
[17]Based on monthly downloads on the huggingface model hub in march 2023

the following research questions:

- RQ1: Is RoBERTa more sensitive to unstructured pruning than BERT?
- RQ2: What impact of using a larger teacher for KD during the pruning of language models?
- RQ3: Can frozen embeddings improve the accuracy of pruned language models?

As part of our experimentation, we release the associated models and the training regimes to aid reproducibility and encourage efficient inference models.
In summary, our contributions are as follows:

- We provide a thorough case study on how to compress a less studied language model, RoBERTa [51], and evaluate performance on a set of seven NLP tasks finding that it is possible to effectively compress a language model without using its original pre-training dataset.
- We demonstrate the impact of varying the size of teachers in KD, freezing embeddings, and variations in learning rates when applied to sparse language models.
- We demonstrate that our compressed models can be leveraged to deliver accuracy of over 91% on the popular SQUAD v1.1 [156] Question Answering Task with nearly three times faster inference than the previous state-of-the-art uses of unstructured sparsity.

3.4.3 Background and Related Work

While many methods to improve model efficiency exist, the same goal generally underpins them: given an original model θ with an accuracy of $acc(\theta)$ and an inference cost of $c(\theta)$ minimize the inference cost. While the methods used for compression can be highly optimized and specialized, they can commonly be used together to deliver massive improvements in inference speeds with minimal losses in accuracy.
Transformer Based Language Models such as BERT [9] and T5 [11] provide contextual language representations built on the Transformer architecture [36] which can be specialized and adapted for specific tasks and domains [133]. Using these models, it becomes relatively easy to excel at a broad range of natural language processing tasks such as Question Answering, Text Classification, and sentiment analysis.
Unstructured Pruning is a compression approach that removes individual weights or groups of weights in a model by applying a mask or setting the weight values to 0. This

compression approach has been broadly studied in computer vision [57], and many methods can remove 70% or more of model weights with little to no loss in accuracy. Models pruned can be 20x smaller in terms of pure model size and, when paired with a sparsity-aware inference engine such as DeepSparse [119], provide 3-5x speedups in inference throughput. Focused on language models, recent work has shown that it is possible to prune models during pre-training with little to no loss in accuracy [66] [138] or during pre-training [137] and transfer to novel domains [26] and datasets.

Structured Pruning is a compression approach that removes fundamental structural components in a language model such as individual attention heads [71] or entire model layers such as transformer encoders [114]. Structural pruning has become one of the most popular methods for inference optimization as it is easy to estimate the speedups and implement.

Freezing Embeddings, as introduced by Devlin et al. [9], involves training the embedding layer of a language model and then toggling the ability to continue to optimize, or not, the values of in the embeddings as training continues.

Knowledge Distillation [76] is a training method where a model is not explicitly a compression method but a training method where a model, called the *student* learns to emulate a *teacher* model which is commonly larger or better performing. The loss extracted from the original training data in KD is augmented or replaced by KL divergence between the student and teacher model.

KD leverages the hardness parameter h to control the mixture of regular and distillation loss (with a higher distillation favoring the KL divergence loss) and a temperature parameter t to control the softness of the distribution.

As applied to language models, the approach has been used to improve the performance of structurally pruned language models resulting in models like DistilBERT [114] and TinyBERT [77].

Quantization precision for the model weights and activations to lower the computational requirements of model execution. While researchers have explored reducing representation to binary representations [158], current hardware limits inference speedups to 8 or 4-bit representations. Quantization can be applied after the model is trained in a one-shot fashion, but this can lead to large losses in accuracy because of rounding errors.

To avoid this pitfall, use quantization-aware training (QAT), where the forward pass of the model is simulated with lower precision. In contrast, the backward pass happens in full precision. By using QAT models, learn to be robust to rounding errors and can result in Quantization having little to no loss in accuracy. In language models, research has produced quantized language models such as Q8BERT [81] and is commonly used in conjunction with structured and unstructured pruning [137] as a way of introducing compounding compres-

sion.
Other approaches, such as early exiting [115] or token pruning [116], have also improved inference efficiency. Still, the inference improvements can be very dataset dependent and, as a result, out of our experimentation frame.

3.4.4 Improving Sparse Transfer Learning

While quantization and pruning have been well studied as applied to language models, work has studied the compression BERT$_{\text{base}}$ or BERT$_{\text{large}}$. Despite existing research, we find that a clear case study that explores how best to create a family of compressed models is lacking, and this work seeks to remedy that. As part of our research, we compare the impact of varying pruning methods, pruning stage, teachers for KD, and freezing portions of the model as applied to the RoBERTa language model.
While performing task-specific compression allows NLP practitioners to broadly adopt improvements in inference efficiency, having access to pre-optimized models is key. We produce a family of 8 general purpose language models, collectively called oBERTa, which progressively get smaller and faster with minimal losses in accuracy.
The oBERTa models leverage a combination of structured and unstructured pruning to provide a set of compressed models which can meet a wide set of latency needs. This compression approach has not been extensively documented nor discussed. Our approach to producing the oBERTA models builds on prior explorations of the combination of compression methods [138] and addresses compression approaches in a staged manner as shown in Figure A.8.
First, we create three structural variants starting with a RoBERTa$_{\text{base}}$ model. The base uses 12 transformer layers, the medium uses 6, and the small uses 3. Following prior work, we select interleaved layers for the 6-layer model and the first, middle, and last layers for the 3-layer model. Then, each of these 3 models is further pre-trained using masked language modeling on the Wikipedia-Bookcorpus text dataset, leveraging KD from a RoBERTa$_{\text{large}}$ teacher. After that, each model is pruned using gradual magnitude pruning (GMP) to a desired sparsity level (90% and 95%) during additional pre-training based on masked language modeling, similar to Zafir et al. [137]. Further background on the RoBERTA model and why we did not prune using the WebText corpus can be found in the appendix.
After pre-training, the sparsity profile is fixed, and models are fine-tuned and quantized on their target task with a small set of variable hyperparameters. Experimentation on the impact of larger teachers, frozen embeddings, and variations in pruning algorithms are discussed in subsequent portions of this work.

3.4.5 Downstream Compression

We explore the impact of introducing unstructured sparsity during task-specific fine-tuning. We repeat each experiment with three different seeds and report the average F1 and Exact Match (EM) metrics in tables 3.10 and 3.11. Following a basic hyperparameter sweep, our baseline RoBERTa$_{\text{base}}$ model achieves a performance of 83.95 EM and 91.13 F1 in the broadly used question-answering benchmark SQUAD V1.1 [156].
We also perform unstructured pruning varying the sparsity 50-95% and the pruning method: GMP and OBS. We prune each model for eight epochs, followed by an additional two epochs to allow the network to stabilize and re-converge. Knowledge distillation is used during training with the dense baseline model as a teacher, hardness set to 1.0 and temperature set to 5.0. Further hyperparameters are in the appendix A.2.7.
Table 3.9 shows the impact of sparsity on BERT$_{\text{base}}$, as reported by previous work. Comparing these results with tables 3.10 and 3.11, we conclude that RoBERTa is more sensitive to pruning than BERT, although RoBERTa$_{\text{base}}$ pruned with OBS remains significantly more accurate than BERT$_{\text{base}}$ for the same level of sparsity.
Table 3.10 shows that pruning RoBERTA$_{\text{base}}$ to 90% with OBS results in a relative drop in F1 of 1.59%, which is three times the relative drop reported for BERT$_{\text{base}}$ with the same pruning algorithm. Moreover, table 3.11 shows that RoBERTA$_{\text{base}}$ becomes very sensitive to pruning with GMP for sparsities above 85%, with the relative drop in F1 increasing almost threefold between 85% and 90% sparsity. We conjecture that RoBERTa is more sensitive to pruning than BERT because the latter is relatively under-trained [51], making the more optimized RoBERTa more sensitive to the loss in expressivity caused by pruning.

Model	Sparsity	F1	Impact
BERT$_{\text{base}}$ [9]	0	88.50	N/A
BERT$_{\text{large}}$ [9]	0	90.9	N/A
RoBERTa$_{\text{base}}$ [51]	0	91.13	N/A
RoBERTA$_{\text{large}}$ [51]	0	94.60	N/A
PruneBert$_{\text{base}}$ [66]	90	84.90	-4.07 %
PruneOFA$_{\text{large}}$ [137]	90	87.25	-1.41 %
oBERT$_{\text{large}}$ [138]	90	87.98	-0.58%
$GMP_{\star\text{large}}$ [159]	90	86.7	-2.03%

Table 3.9: Performance of existing dense and sparse language models on the SQUAD v1.1 Question Answering Dataset

Sparsity (%)	EM	Impact	F1	Impact
50	84.80	1.01%	91.49	0.40%
60	84.64	0.82%	91.33	0.22%
70	84.42	0.56%	91.13	0.00%
80	84.64	0.82%	91.33	0.22%
85	82.89	-1.26%	90.12	-1.11%
90	82.48	-1.75%	89.68	-1.59%
95	79.01	-5.89%	87.05	-4.47%

Table 3.10: Impact of Sparsity introduced by OBS on the F1 and EM scores of pruned RoBERTa models on the SQUAD V1.1 Dataset

Sparsity (%)	EM	Impact	F1	Impact
50	84.90	1.13%	91.46	0.36%
60	84.27	0.38%	90.91	-0.24%
70	83.37	-0.69%	90.30	-0.91%
80	81.64	-2.76%	88.86	-2.49%
85	81.64	-2.76%	88.86	-2.49%
90	76.51	-8.86%	84.90	-6.83%
95	69.39	-17.34%	79.35	-12.93%

Table 3.11: Impact of Sparsity introduced by GMP on the F1 and EM scores of pruned RoBERTa models on the SQUAD V1.1 Dataset

3.4.6 Upstream Compression

Based on our fine-tuning experiments, achieving a high degree of sparsity on the RoBERTA model leads to improvements in performance, but there are greater than expected losses in accuracy. Additionally, such compression is task-specific and non-amortizable, so we explore how best to generate general pruned RoBERTa models. While we eventually apply the winning set of training combinations to all of our variants of oBERTa, we first seek to answer the following questions: Does GMP or OBS perform better during pretraining pruning? Does Freezing the Embeddings during pretraining pruning further improve performance? Does the use of larger teachers further improve performance?
We prune various models while varying individual variables during pretraining to evaluate these questions. We experiment by pruning an oBERTa$_{\text{base}}$ (12 layers) model to 90% and 95% sparsity on all non-embedding layers. All pretraining pruning happens using the Wikipedia-BookCorpus dataset, where we train for five epochs using a learning rate of 5e-5 and a batch size of 256 using 4 A100 GPUS. To evaluate the impact of these models, we evaluate performance on the previously used SQUAD v1.1 question-answering dataset, where we train with a fixed training regime of 10 epochs with a learning rate of 1.5e-4 based on the work of Kurtic et al. We train without KD for each finetuning run with an unpruned

RoBERTa$_{\text{base}}$ or an unpruned RoBERTa$_{\text{large}}$. Details for the hyperparameters used to train all teacher models can be found in the appendix A.2.5.

Comparing the use of OBS vs. GMP as shown in table 3.12, we can see that GMP consis-

	GMP				OBS			
Model	F1	Impact	EM	Impact	F1	Impact	EM	Impact
RoBERTA$_{\text{base}}$	92.18	0.00%	85.59	0.00%	92.18	0.00%	85.59	0.00%
oBERTa 90% No KD	88.34	-4.17%	80.19	-6.31%	87.72	-4.83%	79.35	-7.29%
oBERTa 90% RoBERTA$_{\text{base}}$ KD	88.75	-3.72%	81.35	-4.95%	88.60	-3.88%	81.37	-4.93%
oBERTa 90% RoBERTA$_{\text{large}}$ KD	89.65	-2.75%	83.12	-2.88%	89.63	-2.76%	82.94	-3.09%
oBERTa 95% No KD	86.58	-6.07%	78.81	-7.92%	84.90	-7.90%	76.82	-10.25%
oBERTa 95% RoBERTA$_{\text{base}}$ KD	86.99	-5.63%	79.41	-7.22%	86.14	-6.55%	78.63	-8.13%
oBERTa 95% RoBERTA$_{\text{large}}$ KD	87.60	-4.97%	80.44	-6.01%	86.14	-6.55%	79.84	-6.72%

Table 3.12: Impact on F1 of SQUAD V1.1 of using OBS vs. GMP as the pruning method during pretraining. Impact measures the relative loss in performance vs. the unpruned RoBERTa$_{\text{base}}$ baseline.

tently outperforms OBS. This is the opposite of what is seen when pruning downstream or, in prior work, pruning BERT. We attribute this inversion to not using the web-text dataset and leveraging the Wikipedia-book-corpus instead. We believe that without access to the original training corpus OBS is leading to overfitting of the sparse models, a dataset that is not its intended target.

Evaluating the impact of variations in the hardness of KD as shown in table 3.13, there

	Hardness 0.5				Hardness 1.0			
Model	F1	Impact	EM	Impact	F1	Impact	EM	Impact
RoBERTA$_{\text{base}}$	92.18	0.00%	85.59	0.00%	92.18	0.00%	85.59	0.00%
oBERTa 90% No KD	88.21	-4.31%	80.19	-6.31%	88.34	-4.17%	80.19	-6.31%
oBERTa 90% Base KD	89.19	-3.25%	81.74	-4.50%	88.75	-3.72%	81.35	-4.95%
oBERTa 90% Large KD	90.14	-2.21%	83.51	-2.43%	89.65	-2.75%	83.12	-2.88%
oBERTa-95 No KD	85.82	-6.90%	77.77	-9.14%	86.58	-6.07%	78.81	-7.92%
oBERTa-95 Base KD	86.98	-5.64%	79.23	-7.43%	86.99	-5.63%	79.41	-7.22%
oBERTa-95 Large KD	87.66	-4.91%	80.40	-6.07%	87.60	-4.97%	80.44	-6.01%

Table 3.13: Impact on F1 of SQUAD V1.1 by hardness in KD during pretraining pruning. Impact measures the relative loss in performance vs. the unpruned RoBERTa$_{\text{base}}$ baseline.

is a bit more of a muted set of conclusions. The 95% sparse models perform better with a hardness of 1.0, while the 90% models do better with a hardness of 0.5. Given that our goal is to preserve most of the RoBERTa model without actually using its large dataset, we set our hardness to 1.0 as it keeps the model from explicitly learning the new dataset.

When we evaluate the impact of freezing embeddings during pretraining, as shown in table

	Frozen Embeddings				Trained Embeddings			
Model	F1	Impact	EM	Impact	F1	Impact	EM	Impact
RoBERTa$_{\text{base}}$	92.18	0.00%	85.59	0.00%	92.18	0.00%	85.59	0.00%
oBERTa$_{\text{base}}$ 90% no KD	87.71	-4.85%	79.62	-6.98%	88.21	-4.31%	80.19	-6.31%
oBERTa$_{\text{base}}$ 90% RoBERTa$_{\text{base}}$ KD	89.7	-2.69%	81.74	-4.50%	89.19	-3.24%	83.07	-2.94%
oBERTa$_{\text{base}}$ 90% RoBERTa$_{\text{large}}$ KD	89.59	-2.81%	82.98	-3.05%	90.14	-2.21%	83.51	-2.43%

Table 3.14: Impact on F1 of SQUAD V1.1 with respect to the use of frozen embeddings or not during pretraining pruning. Impact measures the relative loss in performance vs. the unpruned RoBERTa$_{\text{base}}$ baseline.

3.14, we find strong evidence that using frozen embeddings consistently leads to worse performance and, as a result, does not freeze embeddings during our model pruning. Looking at the impact of varying the size of the teacher for pretraining KD as shown in table 3.15, we unsurprisingly find clear evidence that using a larger teacher during pretraining pruning leads to improvements in performance.

Using these experiments, we generate the recipe, which we then use to create the many

	Base Upstream Teacher				Large Upstream Teacher			
Model	F1	Impact	EM	Impact	F1	Impact	EM	Impact
RoBERTA$_{\text{base}}$	92.18	0.00%	85.59	0.00%	92.18	0.00%	85.59	0.00%
oBERTa 90% no KD	88.34	-4.17%	80.59	-5.84%	88.1	-4.43%	80.06	-6.46%
oBERTa 90% Base KD	88.75	-3.72%	81.35	-4.95%	89.22	-3.21%	82.02	-4.17%
oBERTa 90% Large KD	89.65	-2.74%	83.12	-2.89%	89.98	-2.39%	83.14	-2.86%

Table 3.15: Impact on F1 of SQUAD V1.1 with respect variation is the size of the teacher in KD during pretraining pruning. Impact measures the relative loss in performance vs. the unpruned RoBERTa$_{\text{base}}$ baseline.

variants of oBERTa. We pre-train models using KD using a RoBERTa$_{\text{large}}$ teacher with a hardness of 1.0; we do not freeze the embeddings and use GMP to prune.

3.4.7 Experimental Results

Based on the aforementioned experiments, we generate 8 variants of oBERTa, each with a different size and sparsity profile; details can be found in table A.11. Within this table, we report the impact on the model size as measured by the raw and compressed size of the ONNX [18] model file. Embeddings are unpruned, and each layer is pruned to the target sparsity profile independent of the rest of the model. As a result, the overall sparsity profile

[18]https://onnx.ai/

may vary as modules in the network may not be able to reach exactly 90% or 95% sparsity. Using these *inference-optimized* models, we evaluate their *sparse transfer* performance by finetuning these models on their target task using a fixed training regime and minor hyperparameter exploration. For each task, we train them for 10 epochs or 20 (10 of which are Quantization Aware Training), with the longer schedule being reserved for models which are being quantized.

We evaluate performance on a benchmark of diverse NLP tasks ranging from question answering, sentiment analysis, document classification, token classification, and text classification. For question answering, we leverage the SQuAD v1.1 [156] and SQuAD V2.0 [160] datasets. We leverage the SST-2 [161] dataset for sentiment analysis. For text classification, we use the Quora Duplicate Query Detection (QQP) [162] and the MNLI [127] datasets. We leverage the IMDB [163] dataset for document classification and CONLL2003 [164] for token classification.

Looking at performance on question answering as shown in table 3.16 and 3.17. Moving

	Sparse Transfer			Sparse Transfer With Quantization		
model	F1	Recovery	EM	F1	Recovery	EM
oBERTa$_{\text{base}}$	92.15	100.00%	85.78	93.18	101.11%	87.29
oBERTa$_{\text{base}}$ 90\%	90.95	98.69%	84.42	89.46	97.08%	82.61
oBERTa$_{\text{base}}$ 95\%	89.84	97.49%	83.08	89.23	96.83%	81.12
oBERTa$_{\text{MEDIUM}}$	90.37	98.06%	83.84	83.77	90.91%	90.37
oBERTa$_{\text{MEDIUM}}$ 90\%	89.26	96.86%	82.18	88.65	96.20%	81.88
oBERTa$_{\text{SMALL}}$	84.87	92.09%	76.55	84.82	92.05%	76.77
oBERTa$_{\text{SMALL}}$ 90\%	84.66	91.87%	76.18	82.18	92.18%	74.21

Table 3.16: Sparse Transfer performance of the oBERTA family on the SQUAD V1.1 dataset. The sparse transfer was performed over 10 epochs and sparse transfer with quantization over 20. Recovery is based on the relative performance of the unpruned oBERTa$_{\text{base}}$.

to text classification on QQP and MNLI as shown in tables 3.19 and 3.18 Shifting focus to document classification as shown in table 3.20 and sentiment analysis in 3.21 Finally, looking at performance on token classification as shown in table 3.22

3.4.8 Inference Benchmark

To evaluate the performance of our inference-optimized models, we benchmark performance using the popular DeepSparse library version 1.3.2 [19] and an Intel Xeon Gold 6238R Processor. Performance is measured using models that have been *sparse-transferred* to the

[19]pip install deepsparse==1.3.2

	Sparse Transfer			Sparse Transfer With Quantization		
model	F1	Recovery	EM	F1	Recovery	EM
oBERTa$_{\text{base}}$	82.77	100.00%	79.56	85.298	103.06%	82.347
oBERTa$_{\text{base}}$ 90\%	81.33	98.26%	78.27	81.43	98.38%	78.92
oBERTa$_{\text{base}}$ 95\%	77.98	94.22%	74.67	78.09	94.35%	74.82
oBERTa$_{\text{MEDIUM}}$	77.51	93.65%	74.25	78.137	94.41%	75.179
oBERTa$_{\text{MEDIUM}}$ 90\%	76.64	92.60%	73.34	76.24	92.11%	73.51
oBERTa$_{\text{SMALL}}$	71.54	86.44%	67.93	71.591	86.50%	68.087
oBERTa$_{\text{SMALL}}$ 90\%	70.79	85.53%	67.31	69.35	87.79%	65.21

Table 3.17: Sparse Transfer performance of the oBERTA family on the SQUAD V2.0 dataset. The sparse transfer was performed over 10 epochs, and sparse transfer with quantization over 20. Recovery is based on the relative performance of the unpruned oBERTa$_{\text{base}}$.

	Sparse Transfer			Sparse Transfer With Quantization		
model	Accuracy	Recovery	Accuracy(MM)	Accuracy	Recovery	Accuracy(MM)
oBERTa$_{\text{base}}$	87.88%	100.00%	87.57%	88.06%	100.20%	88.01%
oBERTa$_{\text{base}}$ 90%	85.17%	96.91%	84.73%	85.09%	96.83%	84.76%
oBERTa$_{\text{base}}$ 95%	84.32%	95.95%	84.08%	83.73%	95.28%	83.83%
oBERTa$_{\text{MEDIUM}}$	85.29%	97.05%	85.17%	83.62%	95.15%	83.74%
oBERTa$_{\text{MEDIUM}}$ 90%	81.61%	92.87%	81.32%	82.37%	93.73%	81.79%
oBERTa$_{\text{SMALL}}$	80.80%	91.95%	81.55%	81.10%	92.29%	81.51%
oBERTa$_{\text{SMALL}}$ 90%	79.23%	90.15%	79.24%	79.14%	90.06%	79.42%

Table 3.18: Sparse Transfer performance of the oBERTA family on the MNLI dataset. Sparse transfer was performed over 10 epochs and sparse transfer with quantization over 20. Recovery is based on the relative performance of the unpruned oBERTa$_{\text{base}}$.

	Sparse Transfer				Sparse Transfer With Quantization			
model	Accuracy	Recovery	F1	Combined	Accuracy	Recovery	F1	Combined
oBERTa$_{\text{base}}$	91.52%	100.00%	90.09%	88.66%	89.86%	98.18%	88.12%	86.73%
oBERTa$_{\text{base}}$ 90%	91.01%	99.44%	89.47%	87.92%	91.21%	99.66%	89.68%	88.16%
oBERTa$_{\text{base}}$ 95%	90.85%	99.26%	89.21%	87.58%	90.72%	99.12%	89.08%	0.87%
oBERTa$_{\text{MEDIUM}}$	91.35%	99.81%	89.90%	88.44%	91.33%	99.79%	89.80%	88.28%
oBERTa$_{\text{MEDIUM}}$ 90%	90.48%	98.86%	88.85%	87.21%	90.60%	99.00%	89.01%	87.42%
oBERTa$_{\text{SMALL}}$	90.72%	99.13%	89.21%	87.71%	89.74	98.06%	87.99	86.25
oBERTa$_{\text{SMALL}}$ 90%	89.74%	98.06%	87.99%	86.25%	89.73	98.04%	87.98	86.08

Table 3.19: Sparse Transfer performance of the oBERTA family on the QQP dataset. The sparse transfer was performed over ten epochs, and sparse transfer with quantization over 20. Recovery is based on the relative performance of the unpruned oBERTa$_{\text{base}}$.

SQuAD v1.1 dataset and exported to a standard ONNX model format. Benchmarks are run on 4 and 24 cores and a sequence length of 384 with batch sizes of 1, 16, and 64. For each model, the benchmark is run for 60 seconds with a warm-up period of 10 seconds, and we report the throughput (items per second) and the mean, median, and standard deviation per item latency. We present a set of summary statistics of relative speedup across batch sizes

	Sparse Transfer		Sparse Transfer With Quantization	
model	Accuracy	Recovery	Accuracy	Recovery
oBERTa$_{\text{base}}$	95.24%	100.00%	95.44%	100.21%
oBERTa$_{\text{base}}$ 90%	93.64%	98.32%	93.28	97.94%
oBERTa$_{\text{base}}$ 95%	93.48%	98.15%	92.80	97.23%
oBERTa$_{\text{MEDIUM}}$	93.36%	98.03%	94.08	98.78%
oBERTa$_{\text{MEDIUM}}$ 90%	92.24%	96.85%	92.08	96.69%
oBERTa$_{\text{SMALL}}$	93.04%	97.69%	92.52	97.15%
oBERTa$_{\text{SMALL}}$ 90%	91.60%	96.18%	91.28	95.84%

Table 3.20: Sparse Transfer performance of the oBERTA family on the IMDB dataset. The sparse transfer was performed over ten epochs, and sparse transfer with quantization over 20. Recovery is based on the relative performance of the unpruned oBERTa$_{\text{base}}$.

	Sparse Transfer		Sparse Transfer With Quantization	
model	Accuracy	Recovery	Accuracy	Recovery
oBERTa$_{\text{base}}$	94.60	100.00%	92.66	97.95%
oBERTa$_{\text{base}}$ 90%	92.78	98.08%	92.546	97.83%
oBERTa$_{\text{base}}$ 95%	91.51	96.74%	91.399	96.62%
oBERTa$_{\text{MEDIUM}}$	92.89	98.19%	91.06	96.26%
oBERTa$_{\text{MEDIUM}}$ 90%	88.76	93.83%	89.91	95.04%
oBERTa$_{\text{SMALL}}$	90.48	95.64%	91.28	96.49%
oBERTa$_{\text{SMALL}}$ 90%	89.34	94.44%	88.65	93.71%

Table 3.21: Sparse Transfer performance of the oBERTA family on the SST-2 dataset. The sparse transfer was performed over ten epochs, and sparse transfer with quantization over 20. Recovery is based on the relative performance of the unpruned oBERTa$_{\text{base}}$.

	Sparse Transfer			Sparse Transfer With Quantization		
model	Accuracy	Recovery	F1	Accuracy	Recovery	F1
oBERTa$_{\text{base}}$	99.26%	100.00%	95.51%	99.30%	100.05%	95.98%
oBERTa$_{\text{base}}$ 90%	99.11%	99.85%	94.98%	99.05%	99.79%	94.51%
oBERTa$_{\text{base}}$ 95%	98.89%	99.63%	93.32%	98.75%	99.48%	92.61%
oBERTa$_{\text{MEDIUM}}$	99.04%	99.77%	94.39%	99.18%	99.92%	95.15%
oBERTa$_{\text{MEDIUM}}$ 90%	98.79%	99.53%	93.31%	98.73%	99.46%	92.70%
oBERTa$_{\text{SMALL}}$	99.01%	99.75%	94.00%	98.98%	99.72%	94.13%
oBERTa$_{\text{SMALL}}$ 90%	98.47%	99.20%	91.13%	98.25%	98.98%	89.79%

Table 3.22: Sparse Transfer performance of the oBERTA family on the CONLL-2003 dataset. The sparse transfer was performed over ten epochs, and sparse transfer with quantization over 20. Recovery is based on the relative performance of the unpruned oBERTa$_{\text{base}}$.

and inference server configurations as shown in table 3.23. Full inference performance results can be found in the appendix. In analyzing performance, we can see that the introduction

	24 Cores			4 Cores		
Model	BS 1	BS 16	BS 64	BS 1	BS 16	BS 64
oBERTa$_{\text{base}}$	1.00	1.00	1.00	1.00	1.00	1.00
oBERTa$_{\text{base}}$ Quantized	3.10	4.29	4.46	4.09	4.31	4.32
oBERTa$_{\text{base}}$ 90%	3.29	3.80	3.80	3.60	3.34	3.40
oBERTa$_{\text{base}}$ 90% Quantized	4.12	7.05	7.37	7.67	7.59	7.40
oBERTa$_{\text{base}}$ 95%	8.72	4.56	4.65	4.12	3.85	4.37
oBERTa$_{\text{base}}$ 95% Quantized	4.73	8.22	8.56	9.41	9.06	8.68
oBERTa$_{\text{MEDIUM}}$	1.96	1.99	1.99	1.96	1.99	2.02
oBERTa$_{\text{MEDIUM}}$ Quantized	6.20	8.04	8.44	8.43	8.33	8.45
oBERTa$_{\text{MEDIUM}}$ 90%	6.35	7.41	6.84	7.83	6.56	6.72
oBERTa$_{\text{MEDIUM}}$ 90% Quantized	8.94	12.86	13.65	14.99	14.81	14.95
oBERTa$_{\text{SMALL}}$	3.89	3.96	3.99	3.95	3.97	4.03
oBERTa$_{\text{SMALL}}$ Quantized	12.47	14.12	14.08	15.50	15.48	15.70
oBERTa$_{\text{SMALL}}$ 90%	12.22	14.40	14.67	14.05	14.19	14.13
oBERTa$_{\text{SMALL}}$ 90% Quantized	16.21	21.35	23.96	29.77	27.14	27.58

Table 3.23: Latency reduction of the oBERTa family concerning the unpruned oBERTa$_{\text{base}}$ as measured on 24 and 4 cores. Speedup is measured relative to the latency reduction in MS/batch, and BS refers to batch size.

of quantization to a dense model delivers roughly a 4x speedup while quantization on sparse models is closer to 2x. With the introduction of sparsity, 90% leads to slightly under 4x speedup, while 95% leads to slightly over 4x. The impact of structural pruning is roughly equivalent to the size of the as a 6-layer model is two times faster than a 12-layer, and a 3-layer model is four times faster. Since these different compression forms are additive, we can see that a small (3-layer) 90% quantized model performance is 24x (4*4*2). Looking at the variation in a speedup by batch size and the number of cores, we can see that allocating more cores leads to a smaller gap in inference speedup, especially with small batches. From this, we extract that compression is significant when performing streaming inference (batch size 1) on smaller CPUs.
Next, we go ahead and benchmark the oBERTa model performance against existing sparse-transfer models such as oBERT and PruneOFA using the models that have been published [20] in Neural Magic's Sparse-Zoo [21]. We run these models using four cores and a batch size of 1 and compare their speedup (or slowdown) relative to their performance on the SQUAD v1.1 question-answering benchmark. Results can be found in table 3.24 and full results in A.38. Looking at the improvements in accuracy and inference throughput, we find the oBERTa models are 1.3 to 4 times better than models with approximately the same accuracy.

Looking at the competitive results, we find that the oBERTa-* models can deliver signifi-

[20]Since the PruneBERT model is not available in the zoo, we extrapolate numbers using the performance of our oBERTa$_{\text{base}}$ pruned 90% as both models feature 12 transformer encoders and 90% sparsity.

[21]https://sparsezoo.neuralmagic.com/

		Vs. $\text{BERT}_{\text{base}}$		Vs. $\text{BERT}_{\text{large}}$	
Model	F1	Recovery	Speedup	Recovery	Speedup
$\text{oBERTa}_{\text{base}}$ 90%	91.00	102.77%	3.57	100.44%	20.21
$\text{oBERT}_{\text{large}}$ 95% Quantized	90.21	101.87%	3.41	99.57%	19.31
$\text{prunedOFA}_{\text{large}}$ 90% Quantized	89.96	101.59%	2.38	99.29%	13.47
$\text{oBERTa}_{\text{base}}$ 90% Quantized	89.46	101.03%	7.62	98.74%	43.07
$\text{oBERTa}_{\text{MEDIUM}}$ 90%	89.26	98.99%	7.78	96.75%	43.99
$\text{obert}_{\text{base}}$ 90% Quantized	88.00	99.38%	6.96	97.13%	39.37
$\text{oBERTa}_{\text{SMALL}}$ 90%	84.66	90.97%	13.95	88.91%	78.91
pruneBERT 90%	84.90	95.88%	3.57	93.71%	73.82

Table 3.24: Speedups of the oBERTa-family compared to existing published sparse models compared to the performance of $\text{BERT}_{\text{base}}$ and BERT-large. Speedup measures the reduction in latency of MS/batch. $\text{oBERTa}_{\text{base}}$ 90% exceeds the accuracy of $\text{oBERT}_{\text{large}}$ 95% quantized despite being faster, $\text{oBERTa}_{\text{base}}$ 90% quantized performs at the level of $\text{pruneOFA}_{\text{large}}$ 90% Quantized despite being 3x faster, $\text{oBERTa}_{\text{MEDIUM}}$ 90% can outperform $\text{oBERT}_{\text{base}}$ 90% Quantized despite being 30% faster, and $\text{oBERTa}_{\text{SMALL}}$ 90% performs on par with pruneBERT 90% despite being nearly four times faster.

cant gains in performance (F1) relative to speedups. The $\text{oBERTa}_{\text{base}}$Pruned 90% Quantized model achieves an undertaking that nearly matches pruneOFA-large 90% Quantized while delivering about 13x faster inference. Similarly, the $\text{oBERTA}_{\text{SMALL}}$ 90% model provides similar accuracy to PruneBERT despite being over four times faster.

3.4.9 Discussion and Conclusion

Sparse Models require higher learning rates as shown in the tables in A.2.8 sparse language models can be used as general-purpose contextual language models but require the use of a much higher learning rate. When using structurally pruned models like the 6-layer $\text{oBERTa}_{\text{MEDIUM}}$ and the 3-layer $\text{oBERTa}_{\text{SMALL}}$, the optimal learning rate does not vary much within the same task despite the model size. With the introduction of sparsity, the learning rate needs to scale, usually by a factor of five or ten. We find this counterintuitive as the sparse models have fewer parameters to *tune*, so we would expect them to prefer a much lower learning rate. We attribute this to the loss of expressivity in the network driven by its sparsity. Since the network has fewer degrees of freedom to optimize the points which can be optimized moves much more than those that can't.

Larger models compress better as shown by the gap between the sparse and dense models and the gap between models and their quantized counterparts. While 12-layer models can receive 90 or 95 % sparsity and quantization with little to no loss in accuracy, the three and 6-layer models see a much bigger dip. This aligns with Li et al. 2020 [103] in which they demonstrate that larger models are more robust to pruning and quantization. Empirically, this makes sense as the smaller models have *fewer degrees of freedom*, and other portions of the network cannot counteract the reduction in expressivity caused by pruning and quantization.

Bigger Teachers are not always better as shown in the table in A.2.9 the introduction of larger teachers does not always lead to improvements in accuracy. The impact is highly task and model dependent as some datasets like MNLI or QQP see the little impact in using larger teachers, yet datasets like SQUAD or SQUAD v2.0 see large impacts, which are even more pronounced when the student model is smaller.

Frozen embeddings can help, but not always. As shown by A.2.10, the impact of freezing the embeddings is highly task-specific and inconsistent across tasks or models. In question answering, freezing leads to 1-2 point movement for unpruned models and 5-7 points for pruned models. In other tasks like QQP and MNLI, the impact of frozen embeddings tends to be minor or none.

3.5 CONCLUSION AND KEY TAKEAWAYS

In this chapter, we have deeply examined how to apply, transfer, and leverage unstructured pruning to improve the inference efficiency of auto-encoder language models. By using specialized training regimes and compression approaches, we showed that it is possible to introduce large amounts of sparsity in a model without compromising performance. These compressed models can be used in new tasks and domains like an uncompressed model. This ability to compress and transfer demonstrates how using unstructured pruning can be an effective and scalable method of improving the inference efficiency of many NLP models and tasks. In our exploration of aligning dense retrievers, we find the following key takeaways:

When possible, distill. Knowledge distillation is key for compressing models without losing task accuracy. While larger models do not always yield better results, using an uncompressed model as a target representation can result in major improvements in model performance.

Compression approaches are mostly additive. Compression approaches like quantization, structured pruning, and unstructured pruning can lead to large improvements in inference efficiency. When these approaches are combined, the speedups are even greater. However, when compression approaches are combined, realized speedups separately do not lead to fully additive inference speedups. For example, if quantization can give a 4x speedup and pruning can give 3.8x combined, they might lead to 7.2x, which is close but not exactly the 7.8x of perfectly additive systems.

Compress once, transfer forever. Language models compressed during model pretraining can transfer to novel domains and tasks without significant loss in accuracy. Moreover, models are compressed using general domain pretraining transfer to new domains without any further optimizations, and such transferred models can outperform task-specific compression. This leads us to believe it is optimal to create general domain compressed models which can be used without further compression.

CHAPTER 4: ROBUST AND EFFICIENT SEMANTIC RETRIEVAL

4.1 OVERVIEW

Driven by its ubiquity, information retrieval has become a focal area for applying language models. Driven by improvements in representation models and nearest neighbor search methodology, bi-encoders, often called dense retrievers, have become incredibly popular. While these models effectively retrieve content, they are not without fault. Their reliance on language models makes them more expensive than traditional term-based systems and more robust to query noise.

This chapter discusses how representation alignment can be leveraged to improve model inference efficiency and performance on noisy inputs. First, our work examines *KALE*, where representation alignment is combined with structured pruning and asymmetry between models to improve inference efficiency while minimizing losses in retrieval accuracy. Next, our work discusses *CAPOT*, an alignment of representations focused on improving the performance of language models on noisy inputs without the complexities of data augmentation. Both works demonstrate that Kullback-Leibler divergence between two query representations can compress or improve a model with minimal overhead. This alignment can be done after training.

4.2 QUICK DENSE RETRIEVERS CONSUME KALE: POST TRAINING KULLBACK LEIBLER ALIGNMENT OF EMBEDDINGS FOR ASYMMETRICAL DUAL ENCODERS

4.2.1 Overview

In this section, we consider the problem of improving the inference latency of language model-based dense retrieval systems by introducing structural compression and model size asymmetry between the context and query encoders. First, we investigate the impact of pre and post-training compression on the MSMARCO, Natural Questions, TriviaQA, SQUAD, and SCIFACT, finding that asymmetry in the dual-encoders in dense retrieval can lead to improved inference efficiency. Knowing this, we introduce *Kullback–Leibler Alignment of Embeddings* (KALE), an efficient and accurate method for increasing the inference efficiency of dense retrieval methods by pruning and aligning the query encoder after training. Specifically, KALE extends traditional Knowledge Distillation after bi-encoder training, allowing for effective query encoder compression without full retraining or index generation. Using

KALE and asymmetric training, we can generate models which exceed the performance of DistilBERT despite having 3x faster inference.

4.2.2 Introduction

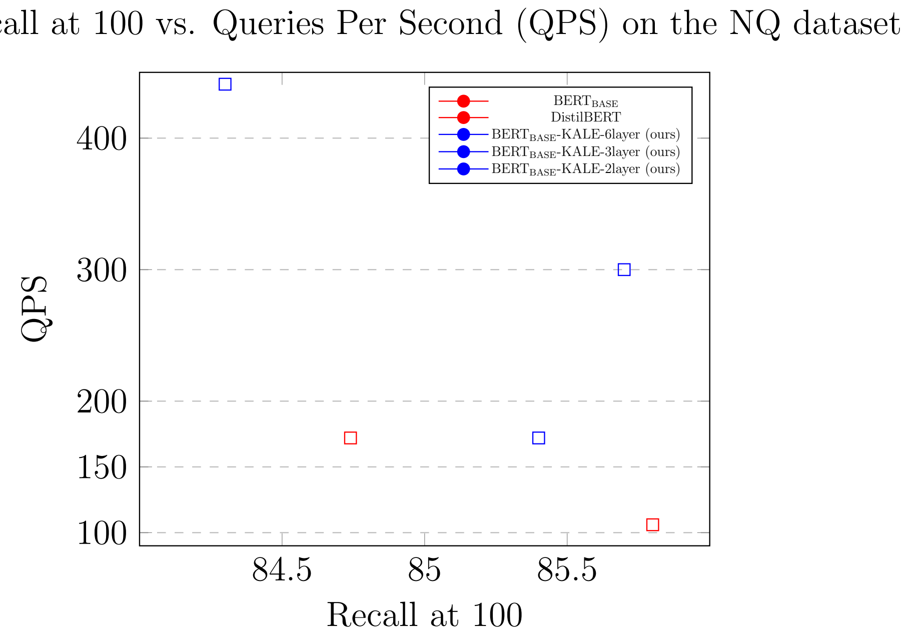

Figure 4.1: Using KALE and asymmetric training on the lead to when measuring QPS vs. Recall at 100 on the NQ dataset. Using Asymmetry and KALE, it is possible to 3x QPS with no loss in accuracy and 4.5x with 1% loss in performance. We calculate QPS as the mean number of queries per second with a batch size of 1 and a max sequence length of 32 on a T4 GPU

A bi-encoder-based retrieval, often called dense retrieval, is a retrieval function that leverages the vector representation of queries and documents as a proxy for relevance. Using two encoders, one for the query and one for the document, the input data is mapped into a common latent space where closeness becomes a proxy for relevance.
Dense retrievers have become increasingly popular due to their ability to capture the semantic relationships between query and document terms. However, bi-encoder-based models can also be computationally expensive, particularly when dealing with large datasets. As a result, there has been a growing interest in methods for compressing these models to reduce their computational complexity without sacrificing performance.
In this section, we explore the role of asymmetry in the size of query and document encoders that leverage language models. Through experiments on several benchmarks, we demonstrate that our approach can significantly reduce the number of parameters in the bi-encoder model without sacrificing performance.

As shown in figure 5.2, the combination of asymmetric bi-encoders and post-training KALE allows for 3x more queries per second (QPS) than an uncompressed bi-encoder with less than 1% loss in accuracy and nearly 5x with less than 2%.
Building on the favorable implications of asymmetry for efficient inference, we introduce a compression mechanism called **K**ullback-Leibler **Al**lingment of **E**mbeddings (KALE). KALE uses a divergence-based alignment of representations to compress models without requiring any form of retraining or index regeneration.
To ground our approaches, we evaluate the effectiveness of KALE and asymmetry on several benchmark datasets and compare the results to existing efficient inference approaches.
The following research questions drive our work:

- Is the performance of dense retrieval methods more driven by the query or document encoder size?
- Is it possible to compress query encoders without retraining and index regeneration?
- How can dense retrieval asymmetry and post-training alignment be leveraged to improve query encoder latency?

It is in answering these questions that we deliver the following contributions:

- We present the first robust study on the role of document-query encoder symmetry, demonstrating that the size of the document encoder dominates performance.
- We introduce and demonstrate the effectiveness of KALE, a post-training compression and alignment approach demonstrating its effectiveness and
- We empirically demonstrate on various benchmarks how Asymmetric Compression can lead to 4.5 better QPS with 1% loss in recall accuracy at 100.

4.2.3 Method

The use of representation models for retrieval begins with a document space d and a query space q, each generated by some model m. Models do not need to share the same initialization, shape, or size, but their representation vectors must share size without some projection. These two models learn a notion of relevance by training to minimize the distance of positive query-document pairs as shown in equation 4.1 where $\mathbf{x}$ is a query vector and $\mathbf{y}$ is a document vector, and $\cdot$ denotes the dot product of the vectors.

$$L = 1 - \frac{\mathbf{x} \cdot \mathbf{y}}{|\mathbf{x}||\mathbf{y}|} \tag{4.1}$$

The query and document encoder models are commonly initialized with a pre-trained language model such as BERT. Then, using pairs of labels for positive relevance scores for queries and documents, the models are trained to minimize the distance between queries and their relevant documents [28]

While it is common practice to initialize the query encoder and document encoder with

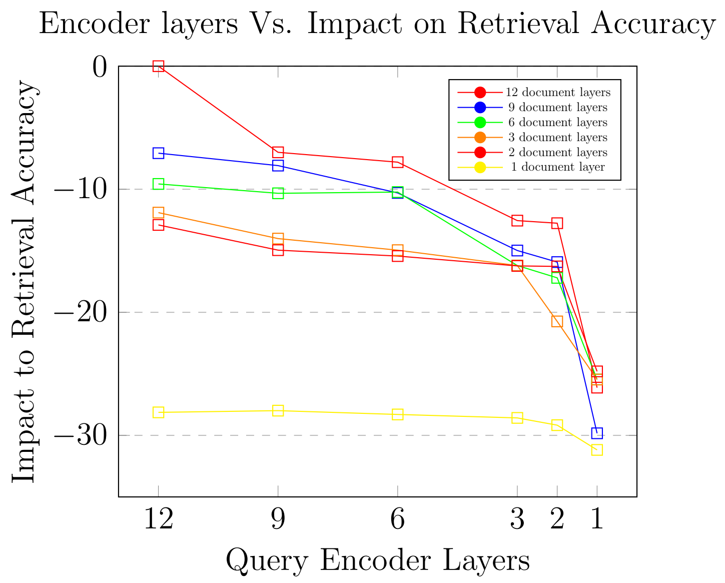

Figure 4.2: Measuring the impact on recall at 20 on the NQ retrieval dataset by varying the number of transformer layers for the query encoder and document encoder

identical language models, this ignores the cost asymmetry of the usage patterns. The document encoder is usually only used once during a large-scale batch generation of the index. Index generation happens in a latency-insensitive environment and can easily leverage many GPUs and large batch sizes to improve efficiency.

The query encoder runs every time a user issues a query, which can be irregular and sporadically. The query encoder responds to each user query independently. Thus, query encoders often use a batch size of 1 and commonly leverage small inference-optimized hardware like the T4 GPU or small CPUs.

Since the document encoder does not run very often, any improvement in latency produces a single fixed gain utterly dependent on the corpus size and index refresh cycle. The query encoder's user-facing nature means latency improvements occur whenever a user queries.

4.2.4 Role of Model Symmetry With Bi-Encoders

The variation in latency sensitivity between the query and document encoder leads to the question: Is there some form of asymmetry between the query encoder and the document encoder that can be exploited? Do the two encoders need to be compressed symmetrically? To answer this question, we explore the impact on the performance of pruning the query and document encoders on the NQ passage retrieval dataset [165]. Using a BERT-base uncased model with 12 transformer encoder layers, we generate structurally pruned models with 9,6,3,2 and 1 layer. We also further pre-train the three and six-layer models using knowledge distillation, represented as 6_{KD} and 3_{KD}, from a 12-layer model on the Wikipedia-book corpus similar to distilBERT [114].
Then, using each of these seven models, we train dense retrieval models on the NQ passage retrieval dataset with variations of query and document models resulting in 72 variants. With each of these models, we generate a full index and evaluate retrieval performance on the development portion of the dataset. We do not tune any parameters to avoid overfitting and to explore asymmetry without overoptimizing. Each model's retrieval accuracy is evaluated with depth 20, 100, and 200 retrieval sets. We compare the impact of varying the encoders to the uncompressed baseline and a distilBERT model (denoted by 6_{db}).

Looking at the impact of symmetric compression as shown in table 4.1, we see that the

Layers *enc*	Top 20	Impact	Top 100	Impact	Top 200	Impact
12	79.86%	0.00%	85.84%	0.00%	88.42%	0.00%
6_{db}	73.88%	-7.49%	84.74%	-1.29%	87.26%	-1.31%
9	73.41%	-8.08%	83.68%	-2.51%	86.51%	-2.16%
6_{KD}	75.04%	-8.08%	85.15%	-0.80%	87.45%	-1.10%
6	71.69%	-8.08%	83.30%	-2.96%	86.04%	-2.69%
3_{KD}	73.32%	-8.08%	83.43%	-2.80%	86.20%	-2.51%
3	66.93%	-16.20%	80.61%	-6.09%	84.49%	-4.45%
2	66.87%	-16.27%	80.42%	-6.32%	83.85%	-5.17%
1	54.96%	-31.18%	71.88%	-16.26%	76.73%	-13.22%

Table 4.1: Impact of Structural pruning before fine-tuning on Retrieval Accuracy on NQ passage retrieval dataset

impact of compression is more pronounced with a small recall set as retrieval accuracy impact at 20 is 3x that of 200. We also observe major accuracy gains by fine-tuning the pruned model with a 4% gap for the 6-layer model and a 6% gap for the 3-layer model for recall at 20.
We find that the document encoder drives retrieval accuracy by looking at the results of asymmetrical training in table 4.2. As shown in figure 4.2, retrieval accuracy is driven by

the document encoder following prior work showing performance out of domain depends on the document encoder [166].

The size of the Document encoder sets the upper bound on a model's performance as a model with 12 layers in the query encoder and 9 in the document encoder performs worse than one with the numbers flipped. The dominance of the document encoder is a logical outcome, as the latent space for queries is simpler than the latent space for documents. As a result, system performance is governed by how well the document encoder can generate representations.

Similar results can be seen with the introduction of fine-tuned three and 6-layer models as shown in B.1. Unsurprisingly, KD-optimized language models outperform non-distilled models, and any asymmetrical variant that leverages a distilled model outperforms the un-distilled variant. Without further optimization, a model with a distilled 3-layer query encoder and a 12-layer document encoder will outperform a model with symmetrical 6-layer models despite being 2x faster.

4.2.5 Inference Benchmarks

To evaluate the impact of structural pruning, we benchmark inference speeds of query encoding while varying the number of transformer layers. We perform benchmarking using an Intel Xeon Gold 6238R Processor and a T4 Nvidia GPU. For each model, we evaluate the performance on encoding 6500 queries with a batch size of one and a max context length of 32. For CPU inference, we evaluate the performance of models using the ONNX library [22], and for GPU inference, we evaluate native Pytorch inference. We repeat each run five times to ensure consistency and report the mean. Summary statistics can be found in 4.3, and full results, including percentile, standard deviation, and confidence intervals, can be found in the appendix C.1.4.

4.2.6 KL Alignment of Embeddings

While training asymmetric models can improve latency, it requires novel training regimes and experimentation, and existing workloads need to regenerate their entire index to take advantage of any inference speedups. Generation of the passage index can take longer than model training [28], which makes regenerating a new index and retraining a model to meet

[22]https://onnx.ai/

$layers_q$	$layers_d$	Top 20	Impact	Top 100	Impact	Top 200	Impact
12	12	79.86%	0.00%	85.84%	0.00%	88.42%	0.00%
9	12	74.27%	-7.00%	84.40%	-1.67%	86.95%	-1.66%
6	12	73.63%	-7.80%	84.27%	-1.83%	86.79%	-1.85%
3	12	69.83%	-12.55%	82.58%	-3.80%	85.35%	-3.48%
2	12	69.67%	-12.76%	82.19%	-4.25%	84.68%	-4.23%
1	12	59.00%	-26.12%	75.37%	-12.19%	81.00%	-8.39%
12	9	74.21%	-7.07%	84.40%	-1.67%	87.06%	-1.53%
9	9	73.41%	-8.08%	83.68%	-2.51%	86.51%	-2.16%
6	9	71.63%	-10.30%	83.05%	-3.25%	85.98%	-2.76%
3	9	67.89%	-14.98%	80.94%	-5.71%	84.79%	-4.10%
2	9	67.15%	-15.92%	80.53%	-6.19%	83.66%	-5.39%
1	9	56.04%	-29.83%	73.35%	-14.55%	78.12%	-11.65%
12	6	72.22%	-9.57%	83.41%	-2.83%	85.84%	-2.91%
9	6	71.61%	-10.33%	83.30%	-2.96%	85.93%	-2.82%
6	6	71.69%	-10.23%	83.30%	-2.96%	86.04%	-2.69%
3	6	66.93%	-16.20%	80.28%	-6.48%	83.96%	-5.04%
2	6	66.12%	-17.20%	80.33%	-6.42%	83.49%	-5.58%
1	6	59.53%	-25.46%	75.37%	-12.19%	79.83%	-9.71%
12	3	70.36%	-11.90%	81.72%	-4.80%	84.60%	-4.32%
9	3	68.67%	-14.01%	80.47%	-6.25%	84.46%	-4.48%
6	3	67.92%	-14.95%	80.06%	-6.74%	83.85%	-5.17%
3	3	66.93%	-16.20%	80.61%	-6.09%	84.49%	-4.45%
2	3	63.30%	-20.74%	78.37%	-8.71%	83.02%	-6.11%
1	3	59.53%	-25.46%	75.68%	-11.84%	80.08%	-9.43%
12	2	69.56%	-12.90%	81.33%	-5.25%	84.49%	-4.45%
9	2	67.92%	-14.95%	80.75%	-5.93%	84.32%	-4.64%
6	2	67.53%	-15.43%	80.33%	-6.42%	83.82%	-5.20%
3	2	66.90%	-16.23%	80.36%	-6.38%	84.24%	-4.73%
2	2	66.87%	-16.27%	80.42%	-6.32%	83.85%	-5.17%
1	2	60.06%	-24.80%	75.29%	-12.29%	79.75%	-9.80%
12	1	57.40%	-28.13%	73.24%	-14.68%	78.56%	-11.15%
9	1	57.51%	-27.99%	73.24%	-14.68%	77.87%	-11.94%
6	1	57.26%	-28.30%	73.52%	-14.35%	78.34%	-11.40%
3	1	57.04%	-28.58%	73.93%	-13.87%	78.39%	-11.34%
2	1	56.57%	-29.17%	73.71%	-14.13%	77.98%	-11.81%
1	1	54.96%	-31.18%	71.88%	-16.26%	76.73%	-13.22%

Table 4.2: Impact of Structural pruning before fine-tuning on Retrieval Accuracy on NQ passage retrieval dataset

changing latency requirements an inefficient experimentation pathway.
Moreover, coupling asymmetry into training makes generating query encoder variants more difficult, as each encoder requires its own index and document encoder.
Motivated by this bottleneck, we introduce Kullback-Leibler **Al**lingment of **E**mbeddings (KALE), a simple method of improving bi-encoder latency by aligning the embeddings of compressed models. KALE is applied after model training and leverages large batch sizes to

layers	size	compressed size	method	QPS	Speedup
12	418	387	GPU	105.852	1.00
9	337	212	GPU	139.494	1.32
6	256	236	GPU	172.338	1.63
3	175	161	GPU	299.45	2.83
2	148	136	GPU	441.422	4.17
1	121	111	GPU	660.64	6.24
12	418	387	CPU	47.278	1.00
9	337	212	CPU	63.24	1.34
6	256	236	CPU	90.386	1.91
3	175	161	CPU	166.012	3.51
2	148	136	CPU	229.666	4.86
1	121	111	CPU	378.534	8.01

Table 4.3: Variation in model throughput according to the serving method and the number of transformer layers. Structural pruning can lead to a six and 8-layer performance increase on GPU and CPU, and pruning a model to 3 layers allows a CPU to offer better inference performance than the GPU.

Layers	KALE	NQ	TriviaQA	MSMARCO	SCIFACT	SQUAD
12	N/A	85.84%	85.84%	88.77%	90.70%	77.16%
9	N	79.97%	79.97%	82.01%	71.07%	71.38%
9	Y	84.90%	84.90%	86.16%	84.87%	73.54%
6	N	68.20%	68.20%	72.68%	22.98%	59.97%
6	Y	83.68%	83.68%	84.68%	85.13%	69.87%
3	N	43.88%	43.88%	11.39%	40.80%	34.42%
3	Y	81.14%	81.14%	82.11%	82.57%	64.37%
2	N	46.90%	46.90%	31.46%	42.66%	37.01%
2	Y	81.94%	81.94%	81.96%	82.57%	63.72%
1	N	12.22%	12.22%	0.00%	3.17%	11.66%
1	Y	71.33%	71.33%	54.36%	66.83%	51.39%

Table 4.4: Impact of structural pruning with and without KALE on Accuracy at 100 across various datasets.

make compression *computationally inexpensive* and *independent of training.* A single V100 GPU KALE can produce a compressed query encoder in less than 5 minutes.

First, a bi-encoder model trains with separate query and document encoders. When training is complete, the document encoder, $e_{document}$, is frozen, and using the query encoder, e_q, a structurally pruned copy, $e_{q'}$, is made. Then, using a sample of queries, the $e_{q'}$ model is fine-tuned to minimize the KL divergence of their query representations as shown in equation 4.2.

$$D_{\mathrm{KL}}(e_{q'} \parallel e_q) = \sum_{x \in \mathcal{X}} e_{q'}(x) \log \left(\frac{e_{q'}(x)}{e_q(x)} \right). \tag{4.2}$$

We explored the use of various distance functions such as cosine similarity, Manhattan distance, and the KL divergence but found little sensitivity in any metric besides KL divergence. We believe this is due to us freezing the document representations, and as a result, cosine distance allows the query embeddings to *drift* more than probability distribution matching methods. To explore this further, we experiment with tuning the temperature for the KL divergence and add a loss scaling factor but find a temperature of one and a scaling factor of ten to be most optimal.
Additionally, we explored using a contrastive loss with random negative and hard negatives mined from the trained encoder but found no positive impact for either method. We leave further exploration of training objective improvement for future work.

4.2.7 Experimental Results

We evaluate the effectiveness of KALE by taking uncompressed $\text{BERT}_{\text{BASE}}$ models and pruning them with and without KALE on various well-established passage retrieval benchmarks. First, models are trained, and indexes are generated using un-optimized $\text{BERT}_{\text{BASE}}$ models. Next, the document encoders are frozen, and the query encoders are structurally pruned to have 9,6,3,2, or 1 transformer layer. Finally, query encoders are aligned using KALE, and we compare the performance of compressed models by comparing the Impact on retrieval accuracy at 20,100, and 200.
To aid reproducibility, each model is trained using the Tevatron [167] [23] library, which makes use of hugginface's transformers to provide a simple interface for exploring neural ranking models. Our experiments focus on the plain $\text{BERT}_{\text{BASE}}$-uncased 12-layer transformer model. While never more capable models exist, the unaltered BERT model is widely used in production workloads, which our experiments seek to emulate.
Our work aims not to produce the highest possible retrieval accuracy for a dense encoder. Instead, our goal is to find the role of asymmetry in bi-encoder models. As a result, we leverage the well-established parameters in all of our experiments without using an advanced methodology like contrastive or curriculum learning.
There are fewer parameters for using KALE, and we deliberately do not optimize on anything but the loss between e_q and $e_{q'}$. In general, higher degrees of pruning require longer training with smaller batches.

Datasets We use a wide variety of standard dense retrieval benchmarks, including MS-

[23]https://github.com/texttron/tevatron

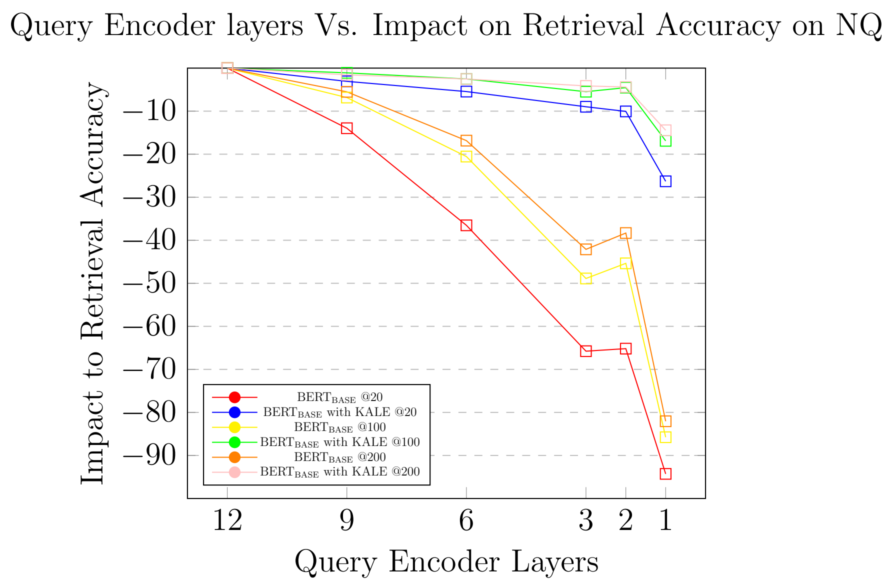

Figure 4.3: Impact of structural pruning with and without KALE on the NQ Passage Retrieval dataset with the recall set sizes of 20,100, and 200. Across datasets, we see a consistent trend where KALE is effective but most effective when the network is heavily pruned and recall set sizes are small. When the model is pruned to 2 or 1 layer with a recall set size of 20, the difference between using KALE or not can be up to 10 times the loss in recall accuracy

MARCO V1.1 [24] [168], NQ Passage Ranking [25] [165], SciFact Passage Ranking [26] [169], TriviaQA passage Ranking [27] [170], and SQUAD Passage Ranking [28] [156].

For each dataset, we evaluate performance by measuring the recall accuracy with retrieval depths of 20,100, and 200. Additionally, for the MSMARCO dataset, we also report MRR@10; for Scifact, we also report NDCG @10 and RR@10.

Computational Experiments Our experimentation on fine-tuning our compressed models uses a 16 GB V100 GPU. Experiments in bi-encoder model training leverage 1 V100 for the MSMARCO and 4 for each other experiment. Due to the vast number of models and datasets we train on, each experiment happens with the same fixed seed.

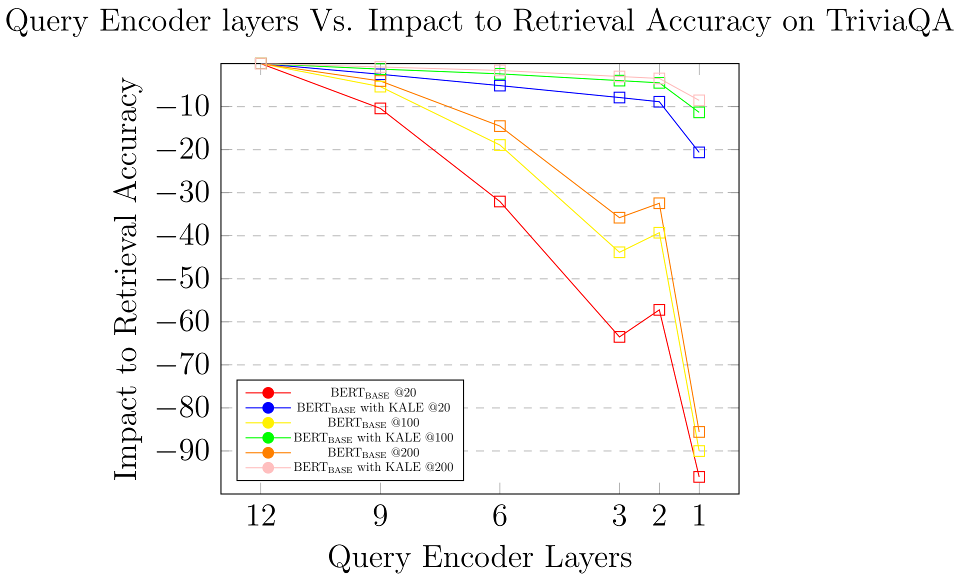

Figure 4.4: Impact of structural pruning with and without KALE on the TriviaQA Passage Retrieval dataset with the recall set sizes of 20,100, and 200. Across datasets, we see a consistent trend where KALE is effective but most effective when the network is heavily pruned and recall set sizes are small. When the model is pruned to 2 or 1 layer with a recall set size of 20, the difference between using KALE or not can be up to 10 times the loss in recall accuracy

4.2.8 Discussion

Evaluating KALE We compare the performance of using KALE for post-training compression in figure 4.7 across the five datasets and see a fairly consistent trend. When the recall set is small, and the query encoders are pruned to a high degree, the impact of KALE is most visible, often driving over 50 improvements in retrieval accuracy. Additionally, using KALE allows the models to have a steady and gradual drop in recall accuracy relative to speedup instead of the sharp drop shown by the regular usage of structural pruning. Without KALE, post-training compression causes a 20-50% loss in retrieval accuracy. With the use of KALE, these losses are cut to 1-10%. In practice, this allows using one or 2-layer encoder models running with CPU-based inference with minor impacts on accuracy.

We also notice a surprising performance improvement between 3 and 2-layer query encoders with and without KALE. This shows the phenomena studied elsewhere: the first and last layers do most of the work [171].

[24]https://huggingface.co/datasets/Tevatron/msmarco-passage
[25]https://huggingface.co/datasets/Tevatron/wikipedia-nq
[26]https://huggingface.co/datasets/Tevatron/scifact
[27]https://huggingface.co/datasets/Tevatron/wikipedia-trivia
[28]https://huggingface.co/datasets/Tevatron/wikipedia-squad

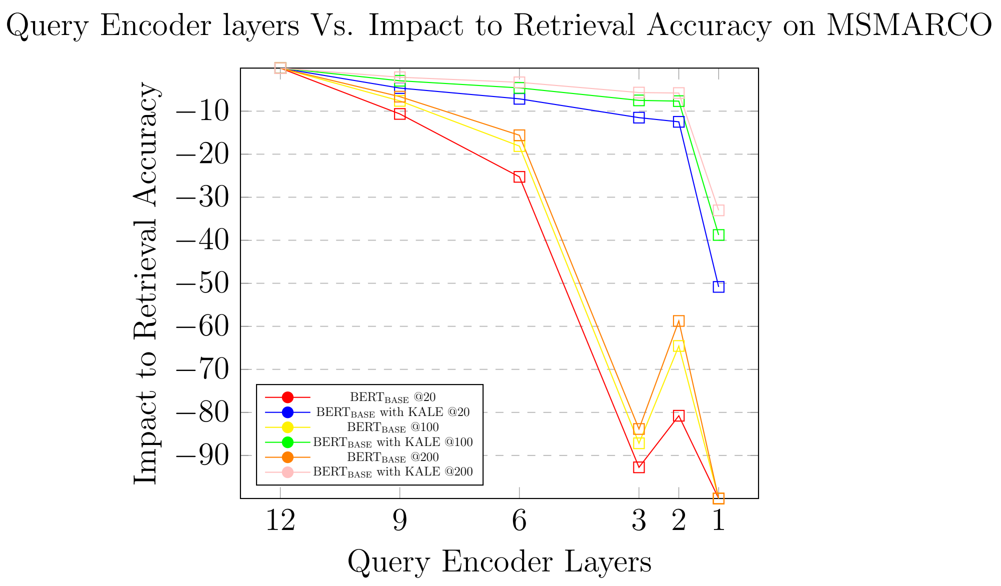

Figure 4.5: Impact of structural pruning with and without KALE on the MSMARCO Passage Retrieval dataset with the recall set sizes of 20,100, and 200. Across datasets, we see a consistent trend where KALE is effective but most effective when the network is heavily pruned and recall set sizes are small. When the model is pruned to 2 or 1 layer with a recall set size of 20, the difference between using KALE or not can be up to 10 times the loss in recall accuracy

Model	Layers	KALE	MSMARCO	NQ	TriviaQA	SQUAD	SCIFACTS
$BERT_{BASE}$	12	N	88.77%	85.84%	85.03%	77.16%	90.70%
$BERT_{BASE}$	6	Y	84.68%	83.68%	83.01%	69.87%	85.13%
$6_{kd} - 6_{kd}$	6	N	88.19%	85.15%	84.96%	71.94%	91.23%
$6_{db} - 6_{db}$	6	N	88.35%	84.74%	84.83%	71.69%	89.37%
$6_{kd} - 3_{kd}$	6	N	86.50%	85.37%	84.04%	70.89%	89.20%
$BERT_{BASE}$	3	Y	82.11%	81.14%	81.67%	64.37%	82.57%
$3_{kd} - 3_{kd}$	3	N	86.13%	83.66%	84.11%	71.98%	89.40%
$3_{kd} - 6_{kd}$	3	N	84.79%	85.76%	83.91%	67.85%	88.63%
$6_{kd} - 3_{kd}$	3	Y	82.95%	83.43%	82.33%	63.77%	90.37%
$6_{kd} - 6_{kd}$	3	Y	86.75%	80.78%	83.48%	64.14%	91.70%
$BERT_{BASE}$	2	Y	81.96%	81.94%	81.23%	67.00%	82.57%
$3_{kd} - 3_{kd}$	2	Y	84.23%	82.71%	83.02%	67.02%	91.33%
$3_{kd} - 6_{kd}$	2	Y	85.57%	84.27%	82.90%	62.75%	88.37%
$6_{kd} - 3_{kd}$	2	Y	83.24%	83.02%	82.13%	62.52%	89.93%
$6_{kd} - 6_{kd}$	2	Y	85.77%	80.39%	83.32%	52.74%	91.93%
$BERT_{BASE}$	1	Y	48.05%	71.33%	75.40%	51.39%	66.83%
$3_{kd} - 3_{kd}$	1	Y	66.69%	77.17%	80.82%	55.62%	76.03%
$3_{kd} - 6_{kd}$	1	Y	72.13%	79.81%	80.23%	52.26%	78.67%
$6_{kd} - 3_{kd}$	1	Y	71.26%	76.57%	78.65%	50.88%	77.07%
$6_{kd} - 6_{kd}$	1	Y	70.70%	74.71%	80.31%	52.74%	77.89%

Table 4.5: Impact of model asymmetry and use of KALE for structural pruning on the Retrieval at 100 accuracies across various datasets.

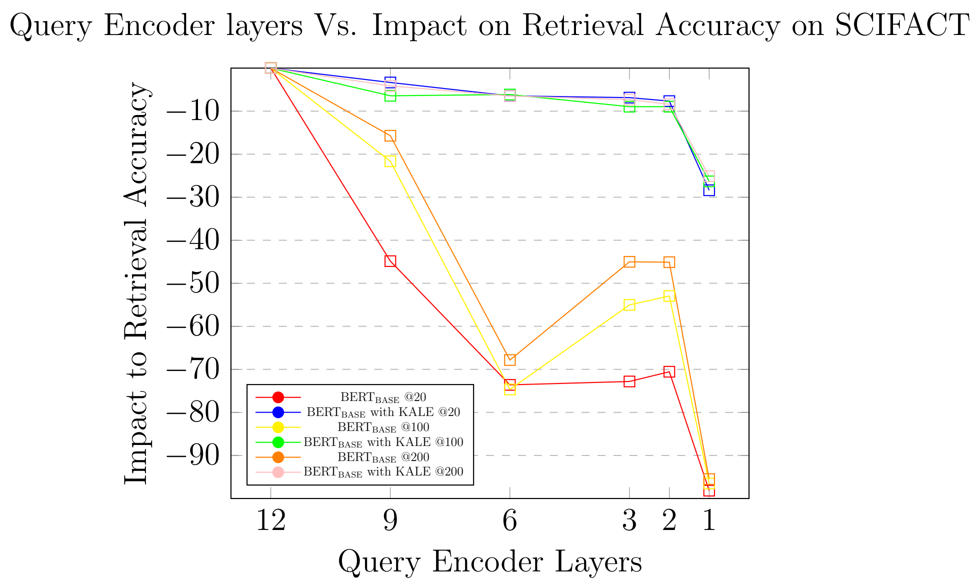

Figure 4.6: Impact of structural pruning with and without KALE on SciFACT, Passage Retrieval dataset with the recall set sizes of 20,100, and 200. Across datasets, we see a consistent trend where KALE is effective but most effective when the network is heavily pruned and recall set sizes are small. When the model is pruned to 2 or 1 layer with a recall set size of 20, the difference between using KALE or not can be up to 10 times the loss in recall accuracy

Aiding Asymmetry with KALE Seeking to optimize compression further, we combine KALE with asymmetrical finetuning and evaluate the results similarly to our earlier experiments. Results on the impact of KALE and asymmetry on the five datasets on the recall accuracy at 100 can be found in table 4.5 where $3_{kd}-6_{kd}$ denotes a three-layer query encoder and six-layer document encoder, $3_{kd}-3_{kd}$ denotes dual three layer encoders. Full results and metrics for each task can be found in the appendix section B.1.4.

First, it is immediately observable that post-training compression via KALE performs worse than models natively designed for that size. We believe this is due to the convergence of the KALE models to have *some distance* from the uncompressed model because of dropout. We experimented with not using dropout in KALE, but model performance quickly suffered.
Looking at the best retrieval accuracy vs. the model speedups shown in figure 4.8, we can see a substantial variation in the impact of compression across datasets. In tasks like SCIfacts, it is possible to get over 4x speedup while improving accuracy, while on tasks like SQuAD, even minor speedups lead to major losses in accuracy. We believe this variation is driven by the relative difficulty of each dataset, where easier tasks are more compressible than harder tasks.
We believe these variations in results highlight the utility of post-training compression meth-

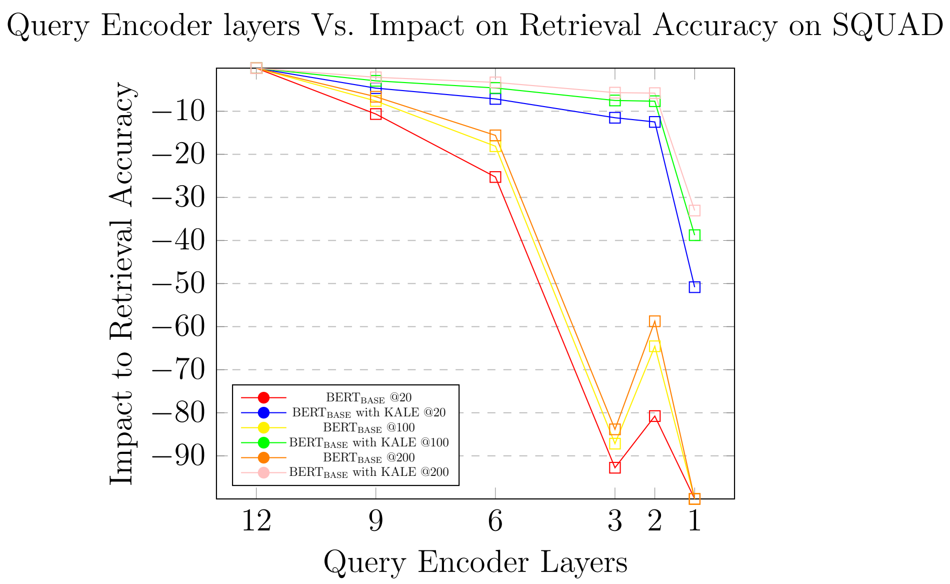

Figure 4.7: Impact of structural pruning with and without KALE on SQUAD, Passage Retrieval dataset with the recall set sizes of 20,100, and 200. Across datasets, we see a consistent trend where KALE is effective but most effective when the network is heavily pruned and recall set sizes are small. When the model is pruned to 2 or 1 layer with a recall set size of 20, the difference between using KALE or not can be up to 10 times the loss in recall accuracy

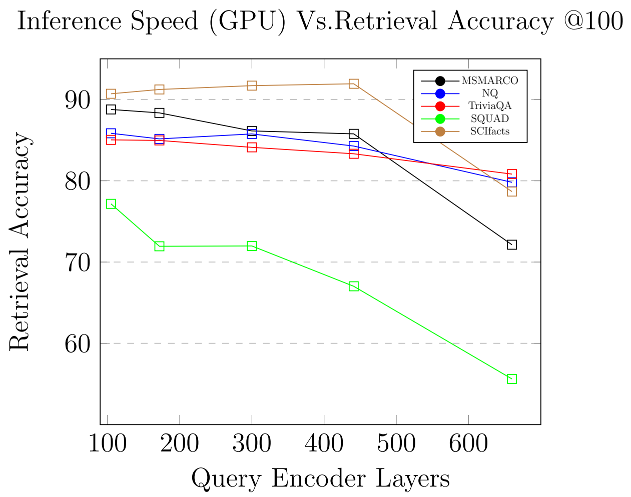

Figure 4.8: The impact on retrieval accuracy of the best combinations of asymmetrical training and KALE across the NQ, MSMARCO, TriviaQA, SQUAD, and SCIfacts retrieval datasets

ods like KALE. Given the task variability in the impact of compression, iteration speed and cost are essential to effectively tuning model inference speed and accuracy.

4.2.9 Conclusion and Future Work

In this work, we have demonstrated how the use of asymmetry between the query and document encoders in bi-encoder models can be leveraged for improved inference efficiencies across CPUs and GPUs. Using our post-training compression framework, KALE, we can compress models up to 6x with little loss in accuracy. Compressing models without regenerating the document index or the document encoder makes it practical to have many query encoders tailored to each use case's latency needs.
In the future, we wish to study how asymmetry in retrieval can be implemented with models which are widely different and may have different hidden sizes, such as using MiniLM for the query model and RoBERTA-Large for the document model.

4.3 NOISE-ROBUST DENSE RETRIEVAL VIA CONTRASTIVE ALIGNMENT POST TRAINING

4.3.1 Overview

The success of contextual word representations and advances in neural information retrieval have made dense vector-based retrieval a standard approach for passage and document ranking. While effective and efficient, dual-encoders are brittle to variations in query distributions and noisy queries. Data augmentation can make models more robust but introduces overhead to training set generation and requires retraining and index regeneration. We present Contrastive Alignment POst Training (CAPOT), a highly efficient finetuning method that improves model robustness without requiring index regeneration, training set optimization, or alteration. CAPOT enables robust retrieval by freezing the document encoder while the query encoder learns to align noisy queries with their unaltered root. We evaluate CAPOT noisy variants of MSMARCO, Natural Questions, and Trivia QA passage retrieval, finding CAPOT has a similar impact as data augmentation with none of its overhead.

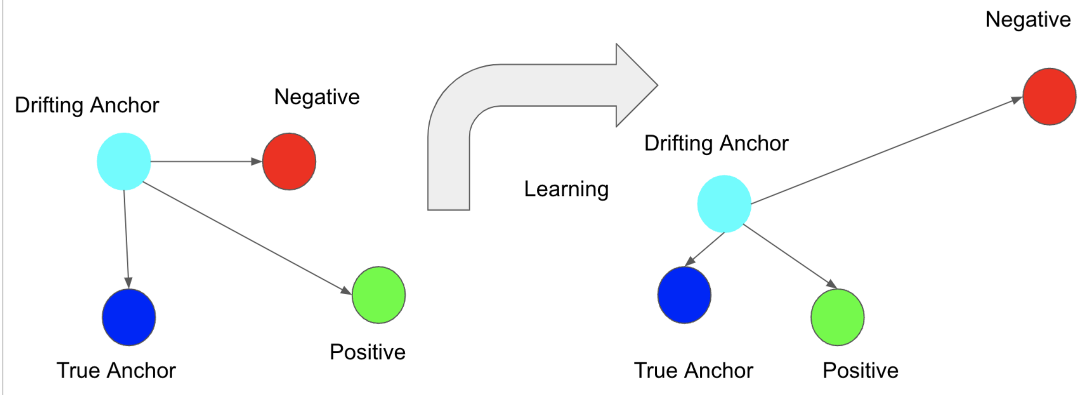

Figure 4.9: To learn a align the representation of queries and their counterparts with noise, a contrastive loss is used. Since the document encoder and its relative relation to the query encoder are frozen, an anchoring vector keeps the aligned encoder from drifting from its original learned representation.

4.3.2 Introduction

Contextual language representations derived from Large Language Models (LLM) have led to impressive improvements in performance across many tasks in natural language processing, such as sentiment analysis, topic detection, and question answering.
In information retrieval, LLM based cross encoders and bi-encoder models have led to major improvements in relevance on benchmarking datasets like MSMARCO [172] and Natural Questions [165] and have been adopted as common backbones for many industrial deployments in search. Unlike traditional term-based search, contextual representations excel at semantic search, which can improve relevance as it matches intent instead of keywords.
While neural methods excel on well-formulated academic benchmarks, performance falters when faced with domain shifts or queries with misspellings or typos. On recent benchmarks like BEIR [89], cross-encoders are more robust to shifts in domain than bi-encoders.
When looking at queries with some noise like typos and misspellings, the same holds [173]. Despite their superior performance on noisy queries and domain shifts, cross-encoder inference requirements make them too expensive to use at scale.
To avoid inference inefficiency of cross-encoder, bi-encoders have emerged as a popular method for retrieval, particularly for candidate generation in multi-stage ranking systems. Bi-encoders leverage query and document models trained to match their representations in

a latent space. Since document representations are query independent, they only need to be generated once, and the inference load is limited to a single run of the query encoder.
Research on bi-encoder has been driven by the availability of large training datasets such as MSMARCO [168], Natural Questions (NQ)[165], and Trivia QA (TQA) [170]. These datasets have allowed deep explorations on how to improve training procedure [31], decrease index size [98], and model efficiency [174]. Despite the tremendous success, these neural methods models are brittle to subtle search domain shifts and minor query formulation variations[175].
While there has been plenty of work that has shown how neural methods are not robust to typos [175] [176] [177] [33] [173] all approaches which improve performance either require a new optimized general model such as CharBERT [178] or require retraining with data augmentation [173]. While effective, both approaches introduce a sizable overhead in dataset generation and augmentation or language model pre-training. Moreover, despite the effectiveness of these two techniques, further study is required to understand the interplay between data augmentation and curriculum learning [179] and topic-aware sampling [180].
Seeking to improve the performance of the query encoder on noisy queries with high effi-

Noising Function	Alteration Type	Original	Alteration
Determiner	Syntactic	who sang waiting for a girl like you	who sang waiting a a girl like you
Synonym	Semantic	Which was the first European country to abolish capital punishment?	Which was the first European country to abolish majuscule punishment?
Lemmatize	Syntactic	who plays young dr mallard on ncis	who play young dr mallard on ncis
Stemming	Syntactic	who recorded the song still the one?	who record the song still the one?
Random Character Swap (RCS)	Surface	big little lies season 2 how many episodes	big litt e lies season 2 how many episodes
Keyboard Character Swap (KCS)	Surface	when did veterans day start being called veterans day	when djid veterans day start being called veterans day
Character Delete (CD)	Surface	when did big air snowboarding become an olympic sport	when did big air snowboarding become an olympic sort
Reorder Word (RW)	Surface	who is the main character in green eggs and ham	who is the main character and green eggs in ham
Back-Translation (BT)	Semantic	what is project charter in project management	What is a project charter in project management
Paraphrasing	Semantic	turkey and china time difference	Time difference between Turkey and China in the middle of the night, depending on the time difference.

Table 4.6: Example of the forms of query noise that we leverage to evaluate how robust bi-encoders are to noise.

ciency possible, we introduce **C**onstrastive **A**llignment **PO**st **T**raining (CAPOT). To avoid complicated dual encoder training regimes, CAPOT assumes that the document encoder and index are immutable and learn an improved query representation without altering existing relations to the index.
As shown in figure 4.9, CAPOT uses a traditional contrastive loss [96] where queries with noise (positive samples) should be closer to the anchor (query without noise) than unrelated queries. Unlike a traditional contrastive loss, CAPOT introduces a notion of an anchoring loss between the unaltered model and the aligned model. As the model learns to group noisy queries with their unaltered roots, we also constrain its ability to alter the representation aligned with the unaltered document encoder.
The main contributions of our work are as follows:

- We introduce CAPOT, a highly efficient fine-tuning method for improving performance on noisy queries without retraining a model or index regeneration.
- We demonstrate that CAPOT is incredibly effective at making the encoder robust, particularly with typos. Using CAPOT approximates the impact of data augmentation without the associated computational overhead.
- We demonstrate that CAPOT is robust enough to prove functional with completely unsupervised data. Using the ORCAS dataset, CAPOT can improve performance without access to the query training distribution.

4.3.3 Generating Noisy Queries

While previous work has studied the impact of minor variations to queries, such as typos and misspellings, query noise is much more diverse. Seeking to expand this understanding, we explore the impact of query alterations that evaluate surface, syntactic, and semantic alterations. To apply noise to a query, we either edit a query to introduce a specific type of noise or rewrite the query to simulate similarly worded intents. Each query that is altered has a notion of its anchor, either a character, word or a group of words, which is selected where noise is applied. To achieve this for each query, a character or word index is randomly selected. Then, noise is applied to the left, right, or at the noising index (replacing the existing index) with equal probability. Example alterations are in table 4.6.

To study the impact of surface-level alterations, we introduce noise in queries by simulating misspellings and typos by swapping, eliminating, or shuffling characters in a query. To understand how models respond to typos or character omissions, we delete a character (DC), inject a random character (RCS), or simulate a keyboard-based typo by injecting a character close to its neighbor on a keyboard (KCS). We swap the indexed word with another word in the query to understand how systems may work when faced with natural shifts in keyword queries.

To study syntactic alterations, we introduce noise that alters the syntax of the query introducing lemmas, stems, synonyms, and determiners using tools from the NLTK toolkit [181]. Synonyms are introduced using NLTK's interface with WordNet [182], and exact synonyms for a single word are introduced. Determiners, affixes that occur with nouns and commonly are not discriminative for search, are introduced similarly to typos to the left or right of noun phrases. Lemma's return words to their canonical root while stemming reduced word inflection using the Porter-stemmer. We select up to five words per query to attempt stemming/lemmatization, but many queries do not have any words which can be stemmed or

lemmatized versions and, as a result, are un-noised.
Exploring semantically similar queries, we leverage paraphrasing, back-translation, and synonyms. To paraphrase, we rewrite queries using a T5 [11] sequence-to-sequence model, which has been fine-tuned on the PAWS [183] dataset. For back-translation, we use OpenNMT's [184] to translate queries from English to another language and then back to English after exploring performance using German, French, Italian, and Spanish to find the German to have the best quality and use only those. It is worth noting that these semantic noising methods are the most likely to alter the true query intent, as seen by the 'hallucinations' in table 4.6 paraphrase alteration.
Using the aforementioned noising approaches, we noise the queries on the MSMARCO [168] [29] [30], Natural Questions [31] [32] [165], and the Trivia QA [170] [33] Passage Ranking datasets.

4.3.4 Baseline Performance

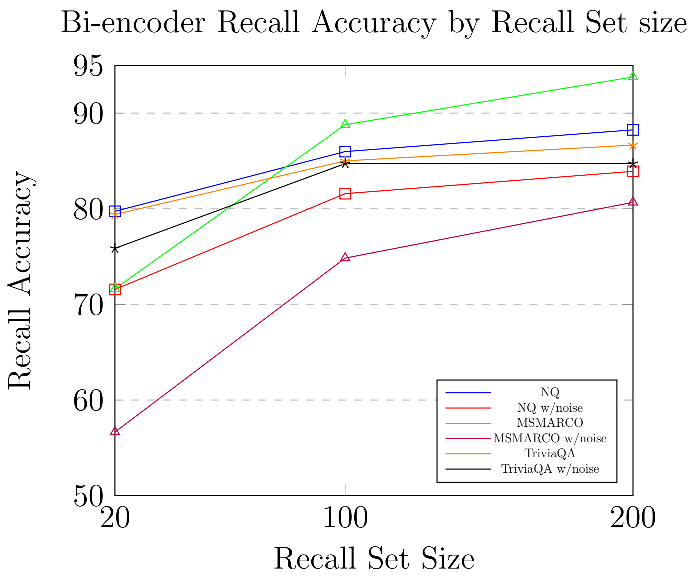

Figure 4.10: Bi-encoder recall accuracy on noisy and non-noisy queries with variations of recall set size and datasets.

In production workloads, bi-encoders are most commonly used for early retrieval. The

[29]https://huggingface.co/datasets/spacemanidol/msmarco-passage-query-variation
[30]https://huggingface.co/datasets/spacemanidol/rewrite-noisy-queries
[31]https://huggingface.co/datasets/spacemanidol/wikipedia-nq-query-variation
[32]https://huggingface.co/datasets/spacemanidol/nq-noising
[33]https://huggingface.co/datasets/spacemanidol/wikipedia-trivia-query-variation

sets they produce are then reranked using a cross-re-ranked Given cross-encoder are more robust to typos [33], our work focuses exclusively on evaluating the impact of noise on the retrieval accuracy of bi-encoders.

To do so, we train a series of task-specific bi-encoders leveraging the open-source bi-encoder-focused library Tevatron [167] with task-specific training parameters found in B.26 on the widely used and studied MSMARCO [168], Natural Questions (NQ) [165] and TriviaQA [170] passage retrieval datasets.

For contextual representations, each encoder uses a pre-trained BERT [9] model for its initialization, and we use separate models for the query and document models. Representations are taken from the unaltered 768-dimension vectors based on the last hidden representation of the CLS token.

For each dataset, we train each model using five different random seeds with fixed optimal hyperparameters, generate seed and task-specific indexes and evaluate the retrieval impact of queries with noise and unaltered routes. We evaluate the impact on retrieval by measuring the impact on retrieval accuracy at k with a $k = 20, 100, 200$.

As shown in figure 4.10 on the impact of averaged noise, our experimental results align with prior research. There is a wide variation of impact as the long, trivia-inspired queries of Trivia QA see minor losses in accuracy compared to the real-world web search queries of MSMARCO, up to 12% the impact. Besides the impact of query type, we also notice the large role recall set size plays on the relative degradation in retrieval accuracy. Across tasks and datasets, increasing the recall set from 20 to 200 decreases the impact on accuracy by about 50%.

Focusing on the impacts of individual types of noise shown in B.2.2, we see that queries with surface alteration, such as typos, see the largest loss. Despite featuring real-world search engine queries with noise, on MSMARCO, there is nearly a 30% loss in retrieval accuracy for queries with typos, dropping from 71% to 41%. On all datasets, queries with character-level alterations see a 50% average higher loss in accuracy than other alterations. This large gap can be attributed to the vocabulary construction method of BERT and BERT-like models, where a minor alteration to a single character can produce large variations in tokenization. In the absence of model optimization, data augmentation, or post-training optimization, the clearest way to make dense retrieval robust to noise is to expand the recall set and allow cross-encoders to re-rank the expanded results.

4.3.5 Incorporating Noise By Aligning Representations

$${}_c(x, x^+, x^-) = \sum_{x \in X} \max(0, \parallel f(x) - f(x^+) \parallel_2^2 - \parallel f(x) - f(x^-) \parallel_2^2 + \epsilon) \tag{4.3}$$

$${}_a(x) = \sum_{x \in X} \max(0, \parallel f(x) - f_a(x) \parallel_2^2 + \epsilon_a) \tag{4.4}$$

$${}_r(x^+, x^-) = \sum_{x \in X} \max(0, -y * (f(x^+) - f(-x) + \epsilon_r) \tag{4.5}$$

$${}_{CAPOT}(x, x^+, x^-) = \sum_{x \in X} \tau_c *_c + \tau_a *_a + \tau_r *_r \tag{4.6}$$

4.3.6 Motivation

A robust query encoder seeks to represent queries with a shared intent in a common latent space such that minor variations in the formulation of the intent lead to similar document ranking. Prior work has shown that data augmentation and typo-optimized models increase model robustness, but it is not without cost.
Data augmentation requires changes to existing training methodologies and complete regeneration of the passage index. Given that the generation of the passage index can take longer than it does to train the model [28] regenerating a new index and retraining a model every time a novel form of noise is discovered is not tractable. Optimized pre-trained models can provide effective modeling solutions. However, given the rapid iteration pace of pretrained language models, making typo-aware variants for each new advance is hard to scale.
Motivated to improve performance without altering the underlying pretrained model or the bi-encoder training regime, we introduce CAPOT, a new methodology for increasing model robustness which is **computationally inexpensive** and **independent of training**. CAPOT works well because it can focus on improving the query encoder and leverages the short nature of queries to scale to large batch sizes.

4.3.7 CAPOT

Contrastive **A**lligment **Po**st **T**raining (CAPOT) is an expansion on traditional contrastive learning focused on making dual encoders robust to noise. The goal of CAPOT is to allow representations of noisy queries to be close to their original on the traditional triplet contrastive loss [96] in 4.3, where f is a query encoder, x is the original query, x^+ is a query

where noise has been introduced, and x^- is a negative query selected at random [34]. We modify 4.3 to scale the role of positive and negative samples using term specific in$\tau_{positive}$ and $\tau_{negative}$ parameters.
While 4.3 allows the query-encoder to represent queries and noisy queries in a similar latent space, it has the unwanted side effect of query representation drifting related to the learned notion of relevance. Without controlling for this drift, a complete collapse in ranking accuracy came at the expense of effective representation of noisy samples. To avoid this, we introduce an anchoring term,4.4, that minimizes the drift between learning a notion of relevance and shared embeddings for queries with noise where f is the noise-robust query-encoder, and f_a is a copy of the unaltered frozen query-encoder, optimized for an existing document encoder and document index.
Seeking to improve performance further, we add a ranking loss as shown in 4.5 between the anchored model f_a and f where the model learns that $f(x^+)$ always ranks higher than $f_a(x)$. While this loss component is not crucial, we can leverage this to improve model performance slightly. 4.3,4.4 and 4.5 are combined to form the CAPOT, 4.6.

4.3.8 Experimental Approach

To qualify the effectiveness, we explore how alignment can improve performance on noisy queries before and after bi-encoder training and compare them to data augmentation. We then compare the performance of the aligned models with unaltered baselines and models trained with DA. Except for models aligned with CAPOT, each experiment requires a complete training run and index generation, which can be quite slow. Each experiment is performed across 5 seeds, and we use the same evaluation metrics previously discussed and report the mean performance over five seeds.
To quantify the ability of post-training alignment, we take the converged baseline models and apply CAPOT to align the model on the training portion of the query set. Once aligned, a model is retrieved on the unaligned, fixed document index generated during our baseline experimentation. Since queries are short and batch sizes can be scaled easily, it's important to note how fast this is. A single 2080ti NVIDIA GPU using CAPOT takes under 60 minutes on the NQ dataset.
To explore if alignment can happen before training, we leverage the ORCAS [185] dataset to generate a corpus of 10 million queries. Using these queries, we create positive and negative noisy samples using the same noising approach discussed in 4.3.3 making a dataset of

[34] We explore the usage of hard negatives mined from nearby query representation but did not find any measurable impact

100 million queries called Noisy-ORCAS [35]. Using these 100m queries, we align the representation of queries and their noisy counterparts using a BERT-base model optimized for masked-language modeling. Given the scale of this dataset, We train for a single epoch on the Noisy-ORCAS corpus using the $\tau_{positive} = 1.0$,$\tau_{negative} = 0.1$,and $\tau_{anchor} = 1.0$ on 4 V100 GPUs with a batch size of 2048. Then, we leverage this optimized model to initialize our unaltered bi-encoder model's training procedure. Then, this model is trained on our datasets and evaluated similarly to the baseline. We refer to models trained this way as(PT), and each usage of PT requires retraining and index regeneration.

4.3.9 Experimental Results

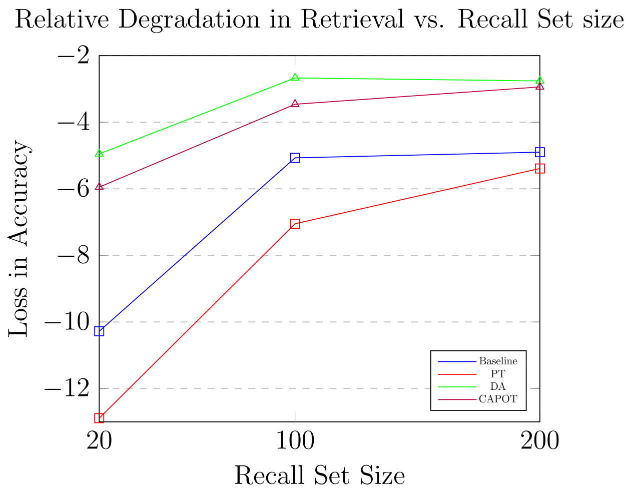

Figure 4.11: Average Relative loss in bi-encoder recall accuracy on NQ by recall set size depth on the baseline, Pretrained Alignment (PT), Data Augmentation (DA), and Contrastive Alignment Post Training (CAPOT) on noisy queries.

As shown in figures 4.11 and 4.12, using CAPOT can improve performance on queries with noise, particularly typos. Moreover, the impact of CAPOT is similar to DA without a training set alteration or index regeneration. CAPOT approach takes advantage of training on only the query encoder and fixing the document encoder. Since queries tend to be short, CAPOT uses a max sequence length of 28 tokens to minimize memory usage, allowing scaling to batch sizes of 2048 on GPUs with 16 GBs. This large batch size means training is rapid and effective. A complete alignment run on the NQ dataset takes one hour on a single V100

[35]https://huggingface.co/datasets/spacemanidol/CAPOT-queries

gpu. At the same time, data augmentation requires 26 hours for training and an additional day for index generation (50 hours overall).

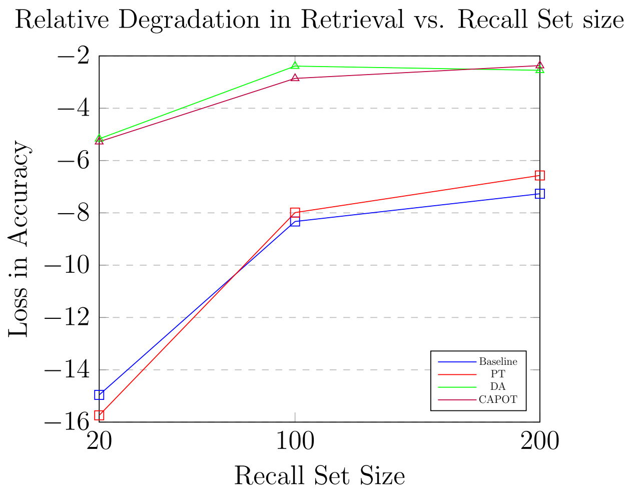

Figure 4.12: Average Relative loss in bi-encoder recall accuracy on NQ by recall set size depth on the baseline, Pretrained Alignment (PT), Data Augmentation (DA), and Contrastive Alignment Post Training (CAPOT) on character-based noisy queries (typos).

Dataset	Regular			DA			PT			CAPOT		
Depth	20	100	200	20	100	200	20	100	200	20	100	200
NQ	-10.28%	-5.07%	-4.91%	-4.95%	-2.67%	-2.76%	-12.89%	-7.05%	-5.39%	-5.95%	-3.46%	-2.94%
TriviaQA	-4.90%	-2.98%	-2.24%	-7.20%	-4.44%	-3.57%	-11.89%	-6.83%	-5.34%	-3.37%	-1.68%	-1.17%
MSMARCO	-20.92%	-33.91%	-30.46%	-43.98%	-28.89%	-16.73%	-46.28%	-36.41%	-28.69%	-22.76%	-16.73%	-14.48%

Table 4.7: Relative degradation in retrieval accuracy at 20,100,200 on NQ,TriviaQA, and MSMARCO. Retrieval accuracy and relative loss across types of noise for unaltered (Regular), Data Augmentation (DA), Pre Training Alignment (PT), and Post Training Contrastive Alignment (CAPOT)

Dataset	Regular			DA			PT			CAPOT		
Depth	20	100	200	20	100	200	20	100	200	20	100	200
NQ	-14.96%	-8.33%	-7.27%	-5.17%	-2.39%	-2.55%	-15.74%	-7.99%	-6.57%	-5.28%	-2.86%	-2.37%
TriviaQA	-8.43%	-4.56%	-3.39%	-7.71%	-4.44%	-3.64%	-14.64%	-8.28%	-5.47%	-3.39%	-1.42%	-0.87%
MSMARCO	-41.68%	-33.91%	-30.46%	-43.98%	-33.70%	-28.69%	-55.58%	-45.47%	-40.95%	-24.58%	-18.40%	-15.95%

Table 4.8: Relative degradation in retrieval accuracy at 20,100,200 on NQ,TriviaQA, and MSMARCO. Retrieval accuracy and relative loss across types of character alteration noise (typos) for unaltered (Regular), Data Augmentation (DA), Pre Training Alignment (PT), and Post Training Contrastive Alignment (CAPOT)

Looking at summary metrics in table 4.7 and 4.8, we can see that the use of pre-training alignment is never optimal and always under-performs un unaltered network. We believe this indicates the importance of introducing noise after training. If introduced prior, the noise will likely be forgotten, hampering learning a proper, relevant representation.

4.4 EXPANDING CONTRASTIVE ALIGNMENT

Seeking to explore the impact of variations in alignment query distribution's role, we explore how well CAPOT works with an alignment dataset that differs from the evaluation. To do so, we explore the impact of using the previously discussed Noisy-ORCAS dataset to align noisy queries for TriviaQA. Given the differences in dataset size, we train for the same number of optimization steps with the Noisy-Orcas data as we do with the regular data.

As shown in figure 4.13, using an unrelated dataset, ORCAS, provides a close approxima-

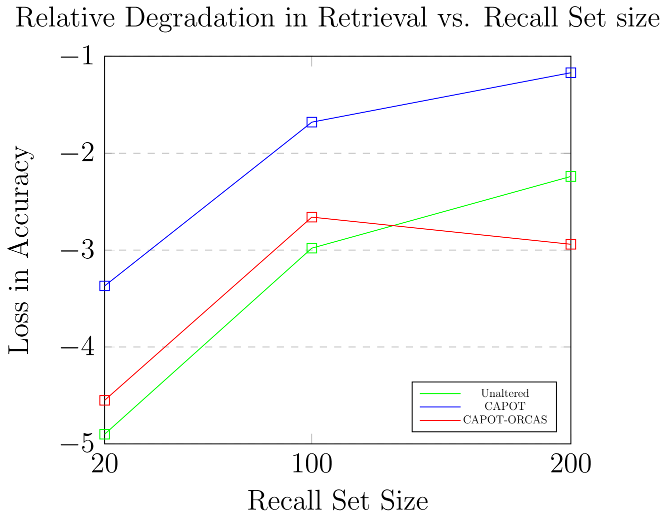

Figure 4.13: Average Relative loss in bi-encoder recall accuracy on NQ by recall set size depth on of unaltered,Contrastive Alignment Post Training (CAPOT) and Contrastive Alignment Post Training (CAPOT) ORCAS on TriviaQA.

tion to using the true query distribution, but it does not always outperform the un-altered baseline, indicated by the performance at 200. We believe this is expected as the true query distribution is a factor in how the query vector manifold is optimized.

4.4.1 Discussion

CAPOT and typos When queries have typos, CAPOT is a computationally efficient method of improving performance as the relative gap between unaltered and aligned is greatest on alterations like character deletion, keyboard character replacement, and random character replacement. We attribute this impact to the relative importance of our alignment dataset's character level alterations. Three out of the ten methods focus on learning alignments based around minor character shifts, and as a result, the performance optimizes there to the detriment of other forms of noise. CAPOT is much less effective in improving the relevance of minor syntactic shifts, such as lemmatization or stemming leading to marginal improvements over the unaltered baselines. We attribute this to the already smaller gap on syntactically altered queries, which on datasets such as TriviaQA have less than 2% impact.
CAPOT and Retrieval Set Depth demonstrates that CAPOT, like DA, sees the highest impact when the recall set size is small. On the NQ, the gap between CAPOT and the baseline at 20 is nearly 10% which narrows to 3% at 200.
Limitations of contrastive alignment While effective, contrastive alignment has a non-negligible impact on the retrieval accuracy of unaltered queries. As shown in table B.40 on non-noisy queries, data augmentation incurs no loss in accuracy, yet CAPOT incurs 2.5%. This is a fundamental issue because the alignment of embeddings causes minor variations in representations that have actual implications on retrieval accuracy. We believe that the use of larger datasets could, such as the web search logs used by the Generic Intent Representation of query vectors [186], could improve this.
Poly Encoding using alignment-based optimization leads to novel retrieval methods which allow for fixed index, constrained optimizations tailored to specific types of noise or deficiencies in retrieval. Novel noise-optimized encoders can be deployed in parallel without additional index generation. Given the prevalence of bi-encoders as candidate set generation tools, CAPOT, unlike Data Augmentation, can generate many targeted query encoder variants which share a document representation. As shown in figure 4.14, instead of seeking a single query encoder that learns all surface and semantic forms of query representations, alignment approaches can create many encoders tuned to various goals.

4.4.2 Summary and Future Work

This work studies how to improve bi-encoder performance on noisy queries efficiently. By extending the contrastive triplet loss with an anchoring loss, CAPOT can be used as an approximation for data augmentation without the associated computational overhead. By

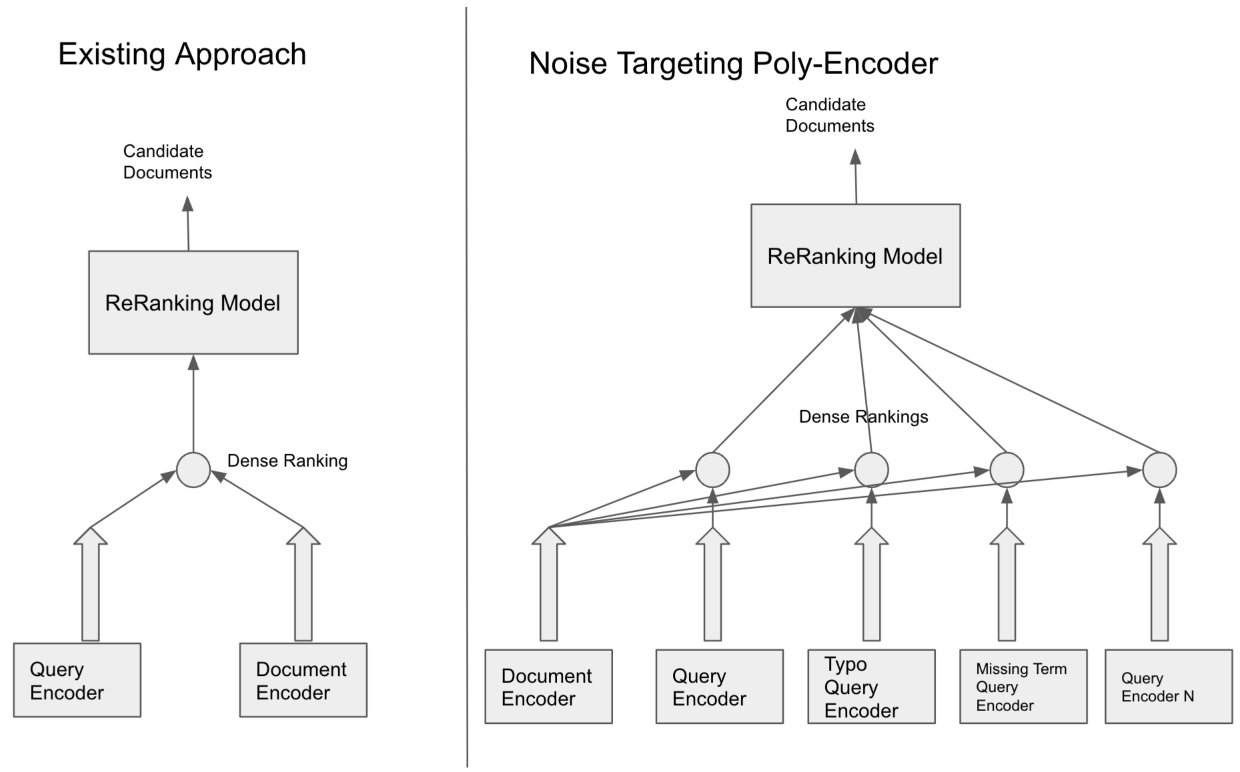

Figure 4.14: Proposed poly-encoder architecture using noise-targeted query encoders optimized with CAPOT

avoiding retraining and alternating the training corpus, CAPOT can significantly improve recall accuracy with 20 times less computational overhead.
In future work, we wish to study how representation alignment approaches can be used with compression approaches such as distillation, pruning, and quantization.

4.5 CONCLUSION AND KEY TAKEAWAYS

In conclusion, in this chapter, we have explored the utility of representation alignment for the compression and robustification of bi-encoders for information retrieval. Using specialized and general task formulations, we can improve the inference efficiency by up to 8x and the performance on noisy queries by 4x with minor computational overhead. By aligning the representations, query encoders can be modified, improved, and compressed without needing to regenerate document representations.
In our exploration of aligning dense retrievers, we find the following key takeaways:
Batch size matters. The maximum batch size often slows down training methods. Since training representation systems in isolation focus models on more uniformity of inputs, mem-

ory utilization decreases, allowing batch sizes to scale. By scaling representation alignment to batches of 2048, compression can take less than 5 minutes on a V100 GPU and be optimized quickly.

Exploit model symmetry. In bi-encoder-based retrieval, some components are computer once offline, while others are continually online. Using this asymmetry in inference costs and the higher impact of compression of the document encoder allows for an asymmetric bi-encoder, improving latency profiles and retrieval accuracies.

CHAPTER 5: SCALING MULTI-LINGUAL CLASSIFICATION AND ABSTRACTIVE SUMMARIZATION TO WEB-SCALE WORKLOADS

5.1 OVERVIEW

As the use of large language models proliferates and use cases mature, there is a clear need for improvements in inference efficiency. Model efficiency and accuracy improvements lead to clear wide-scale deployments requiring fast and cost-effective inference. Building on the previous few chapters, we combine many compression approaches we discussed with industrial workloads with susceptible latency requirements. This chapter will discuss approaches and challenges in speeding up multi-lingual multi-task classification and abstractive summarization for production workloads.
First, we examine the role of multi-linguality and multiple tasks in compressing text classification workloads for experience management. While large, monolingual language models will likely deliver impressive classification accuracy across languages and domains, executing inference and maintaining many monolingual models can be overly complex and cost prohibitive. Using multi-task and multi-lingual models and quantization and knowledge distillation to deliver 44% cost reductions with minor losses in accuracy.
Next, we discuss *asymmetry of sequence-to-sequence models* and how asymmetrically structured pruning can be leveraged for significant improvements in inference speed with little to no loss in accuracy. Using the Shrink Then Fine-tune (SFT) paradigm, we find that pruning a sequence-to-sequence encoder leads to significant losses in summarization accuracy with minor inference efficiency gains. Conversely, decoder pruning leads to substantial improvements in inference efficiency with negligible impacts on summarization accuracy.

5.2 COMPRESSING CROSS-LINGUAL MULTI-TASK MODELS AT QUALTRICS

5.2.1 Overview

Experience management is an emerging business area where organizations focus on understanding the feedback of customers and employees to improve their end-to-end experiences. This results in a unique set of machine learning problems to help understand how people feel, discover issues they care about, and find which actions need to be taken on data that differ in content and distribution from traditional NLP domains. In this paper, we present a case study of building text analysis applications that perform multiple classification tasks efficiently in 12 languages in the developing business area of experience management. To scale

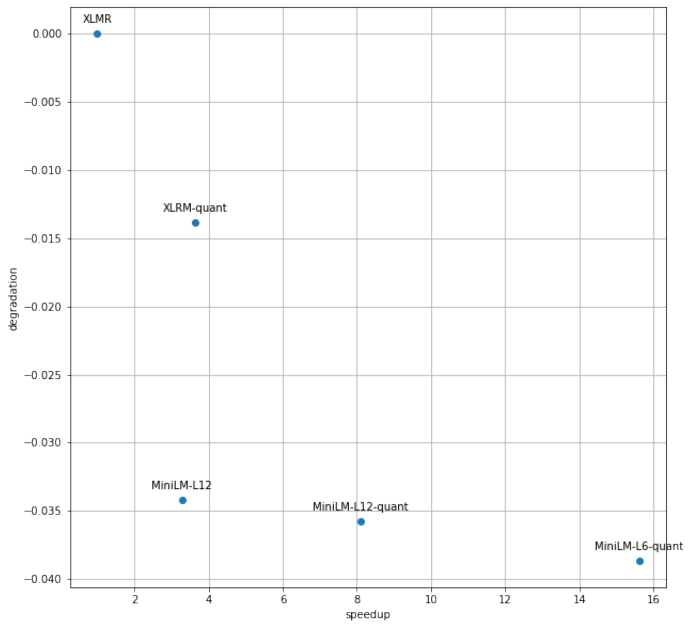

Figure 5.1: Relationship between model speedup and relative degradation to performance for multi-lingual multi-task classification.

up modern ML methods on experience data, we leverage cross-lingual and multi-task modeling techniques to consolidate our models into a single deployment to avoid overhead. We also use model compression and distillation to reduce overall inference latency and hardware cost to the level acceptable for business needs while maintaining model prediction quality. Our findings show that multi-task modeling improves performance for a subset of experience management tasks in XLM-R and mBert architectures. MiniLM achieved the best compression/performance trade-off among the compressed architectures we explored. Our case study demonstrates a speedup of up to 15.61x with 2.60% average task degradation (or 3.29x speedup with 1.71% degradation) and estimated savings of 44% over using the original full-size model. These results demonstrate a successful scaling up of text classification for the challenging new ML area for experience management.

5.2.2 Introduction

Experience management enables businesses and organizations to effectively adapt to actionable customer and employee feedback. Understanding and managing a customer or employee experience requires analyzing a combination of survey questions, social media, and other sources of experience data together to derive insight. Deriving insight requires effectively predicting the feelings, emotions, and requested actions from feedback in a scalable way. To accurately predict these insights, we leverage state-of-the-art pretrained language models. However, many of the best pretrained models require significant resources to deploy at scale. For example, the best-performing models for tasks such as semantic similarity of sentences [187] can have hundreds of billions of parameters [188]. Even more pedestrian (in terms of size) models such as BERT-base [189] can still be relatively expensive and latency-prone for a typical business use case, especially without specialized hardware or accelerators. Leveraging model compression is one way to achieve high prediction accuracy and scale-up. While substantial literature on model compression exists, it cannot be easy to sort through all the methods and evaluate them on a case-by-case basis. Our contribution is a specific case study evaluation of model compression methods for Qualtrics models on experience management data and its unique challenges. In this work, we are particularly interested in building efficient text classifiers, a fundamental problem in the experience management domain. Indeed, unstructured data constitutes more than 80% of the experience management data. As such, analyzing text data across dimensions such as sentiment, emotion, actionability, effort, intent, topic, urgency, and toxicity is one of the most foundational challenges in this emerging space. We share details about what worked and did not, which can benefit the industry as others adopt model compression in their organizations for their use cases. This is particularly timely in our current market as many companies in emerging business areas want to reduce costs. Model compression is an effective way to minimize ML inference costs, both financially and environmentally.

Motivating Constraints: Engineering Overhead, Cost, Latency Our goal in pursuing this compression and consolidation work was to reduce overall model hosting costs while preserving model quality. Two areas we are focused on are reducing the burden for engineering support of hosting our new models in production and the direct cost and latency of the models themselves.

Since we deploy and support our models in a microservices framework, each model lives behind a specific model endpoint or service. Each model has a fixed cost for the base capacity and a variable cost for the elastic ability. If we use single-task monolingual models, this results in needing support in the production of a specific service-per-task language pair. Similarly, for single-task NLP models, the encoder, which can account for 90+% of the computation for classification models, must run for each task, regardless of how similar it is between functions.

In contrast, a multi-task cross-lingual model consolidates this repetitive computation and removes the instance hosting overhead for additional languages. For this reason, we focused on the ability to support multiple tasks per model and a cross-lingual model.

In addition, by developing smaller models, we hope to achieve reduced latency for runtime needs while reducing costs by providing flexibility to deploy on less costly hardware.

5.2.3 The Tension Between Model Consolidation and Compression

An exciting tension arises as we combine multiple models into a single multi-task cross-lingual model and reduce the model's size and capacity. While prior work has also looked at these different facets of model consolidation and compression in isolation [114], [190]–[196], in this work, we investigate how these approaches work together to consolidate and compress a model, and how that impacts model performance on the target tasks.

We cannot analyze this tension for all NLP tasks in general, but here we present evidence for understanding the trade-offs for specific cases relevant to work at our company. These results can inform future theoretical work and more practical applications at other organizations.

5.2.4 Cross-Lingual Multi-Task (XLMT) Model Compression Methods

As described above, we are motivated to consolidate task and language support into a single cross-lingual multi-task (XLMT) model and ,simultaneously pursue a compressed version of that model to reduce capacity and make the model faster and less expensive to run.

5.2.5 Cross-lingual Modeling

There has been a strong movement towards multi-lingual and cross-lingual models. One of the first multi-lingual BERT models was "multi-lingual BERT" (mBert), from [189], which extended "monolingual BERT" by training across a dataset with multiple languages represented. Cross-lingual modeling (XLM), presented in [197], further improved over multi-lingual modeling by introducing additional cross-lingual pretraining tasks, and XLM-Roberta (XLM-R) [198] developed a robust cross-lingual model using techniques from Roberta [199] and showed better performance beyond previous multi-lingual and cross-lingual models.

This work shows results using the mBert and XLM-R pretrained models on which we build our task-specific classifiers. In the original paper [198], the authors showed a decrease in model performance as more and more languages were introduced. We explore the effect of training on monolingual vs. cross-lingual settings and how it impacts our combined model performance.

5.2.6 Multi-task Learning for NLP

Multi-task learning (MTL) can merge tasks into a single model and improve task performance by sharing common layers. For instance, [200] proposed an architecture that shares the same character embedding layer showing effective results for low-resource settings. Other types of MTL include hierarchical architectures, such as [201] where separate tasks are learned and combined using a final attenuation layer and [202] where the first task output feeds into a second task in sequence. In this work, we explore how combining multiple tasks into a single cross-lingual model impacts performance on each task individually. Our approach leverages a standard base model with multiple task heads. The multi-task multi-class classification loss function we use consists of a simple sum of cross-entropy losses,

$$L_{\text{MT}} = \frac{1}{N}\sum_{t=1}^{T}\sum_{i=1}^{N^t}\left[-\sum_{c\in C_t}\left(\ell_{i,c}^t \log p_{i,c}^t\right)\right], \tag{5.1}$$

where $N = \sum_{t=1}^{T} N^t$ is the total number of data points from T tasks and N^t is the number of data points for the t-th task. C_t is the number of classes for task t. $\ell_{i,c}^t$ is either 0 or 1, indicating whether class label c is the correct classification of the i-th data point from the t-th task, and $p_{i,c}^t$ are the corresponding predicted probabilities.

5.2.7 Model Compression

Knowledge Distillation popularizes knowledge distillation (KD) [203] and aims to create smaller models which approximate the performance of the larger models by teaching the smaller model (student model) to emulate the larger model (teacher model). The original approach used the final layer of logit-based knowledge distillation. The concept is to minimize the distance (i.e., KL divergence loss function) of logit output (last layer) between teacher and student models. Later work, including many applications in NLP, introduced variations on this idea, including [114], which applied a combined loss, including masked language modeling loss, cosine distance loss, and KL divergence loss to reduce BERT model size. More generally, we can align the intermediate features between the teacher and the student models rather than just the final layer, such as [194], which uses many intermediate layers for distillation. MiniLM was introduced in [190] using self-attention distribution transfer and self-attention value-relation transfer to achieve competitive performance in monolingual

and multilingual models.
In this work, we have primarily investigated distilling using the task-specific logits produced by the final layer. Exploring additional intermediate representation, distillation is left to future work to improve performance in the most miniature models we tested potentially. Focusing on the last layer results in the following modified loss:

$$L_{\text{MT-KD}} = \frac{1}{N}\sum_{t=1}^{T}\sum_{i=1}^{N^t} \left[-\left(\sum_{c\in C_t} \ell_{i,c}^t \log p_{i,c}^t\right) + \alpha F^2 \left(\sum_{c\in C_t} \hat{q}_{i,c}^t \log \frac{\hat{q}_{i,c}^t}{\hat{p}_{i,c}^t}\right)\right], \tag{5.2}$$

where $q_{i,c}^t$ is the teacher model prediction of the i-th data point from the t-th task, $\hat{q}_{i,c}^t = \frac{\exp(q_{i,c}^t/F)}{\sum_j \exp(q_{j,c}^t/F)}$ is the temperature modified teacher prediction, $\hat{p}_{i,c}^t = \frac{\exp(p_{i,c}^t/F)}{\sum_j \exp(p_{j,c}^t/F)}$ is the temperature modified student prediction, F is the temperature parameter [203], and α is the teacher coefficient term controlling the relative impact of distillation to the label loss.

Structural Pruning In [58], the author introduced a notion that neural networks can be compressed by removing entire sections without significantly impacting accuracy. Structural pruning compresses networks by removing fundamental structural components like attention heads, neurons, and even transformer layers and leverages KD to limit model degradation. While most previous work in compression has focused on monolingual models, there is also a growing body of work around multilingual and cross-lingual model compression [190]–[193], [204]. We focus on two specific compressed architectures, MiniLM [191] and XtremeDistil [192], and compare them in our use case. Ultimately we found MiniLM to be the most effective at learning our specific set of tasks.

Quantization Quantization enables a reduction in model size and memory footprint while also potentially increasing inference speed. Here we consider integer quantization, in which the precision is reduced from a 32-bit floating point to an 8-bit integer. Quantization can be done during training, known as quantization aware training (QAT), to minimize degradation, or after training, known as post-training quantization (PTQ), to compress an already trained model. [205] shows that by leveraging QAT, their "Q8Bert" quantized model was able to match the performance of the base BERT model on various NLP tasks.

In this work, we explore combining quantization via QAT with structural pruning to reduce the model size further while maintaining good model performance.

5.2.8 Experimental Results

Our core set of results is developed around a multi-task cross-lingual model developed internally at Qualtrics to help build understanding around customer feedback. The model handles three separate but related multi-class classification tasks on text input; we refer to these tasks throughout this paper as Task-1, Task-2, and Task-3. They refer to three text classification tasks our group actively uses and develops, with similarities to models such as sentiment or toxicity prediction [202]. Each task is implemented as a sequence classification task where the input is direct customer feedback. Task-1 is a multi-class classification with six labels, Task-2 is a multi-class classification with four labels, and Task-3 is a multi-class, multi-label sequence classification with nine classes, and each class has independent binary labels.

Our experiments explore the relationship between knowledge distillation, multi-task modeling, quantization, and multilingualism. We do not seek to provide a complete understanding of how each axis impacts the outcomes. Instead, we strive to find the optimal way to optimize the performance of pruned and quantized models by exploring the impact of multi-lingual fine-tuning, variations in knowledge distillation, and task-specific teachers.

5.2.9 Dataset and Compressed Architecture Selection

Our dataset consists of internal customer experience data across multiple industries. The data has been fully anonymized and aggregated and is used with permission. This process protects customer data privacy and ensures data from any specific industry or company is not over-represented or identifiable. The resulting text dataset comprises 257k text documents across 16 languages labeled for Task-1 and 127k text documents across 12 for Task-2 and Task-3. A description of the task types, number of labels, and labels can be seen in Table 5.1. This experimental data is similar to the production datasets used in our production modeling system.

For modeling, we primarily use PyTorch [206] and the Transformers library [207]. For model quantization, we used the SparseML library [124], [208].

Instead of developing our target architecture, we leverage existing cross-lingual models from the literature as a first approach to model compression. After a review of the literature, we settled on experimentation around two cross-lingual models, XtremeDistil [204], [192] and MiniLM [190], [191]. We summarize the characteristics of the architectures evaluated in Table 5.2, where the most miniature model considered was 6 layers and 22M parameters.

To further narrow to a single cross-lingual model, we experimented using a subset of our

Task-1 (Multi-class)		**Task-2** (Multi-class)		**Task-3** (Multi-label)	
# Samples		# Samples		# Samples	
T1-L1	93738	T2-L1	83545	T3-L1	33463
T1-L2	70218	T2-L2	36018	T3-L2	21562
T1-L3	38786	T2-L3	4317	T3-L3	22556
T1-L4	26359	T2-L4	3198	T3-L4	1090
T1-L5	18792			T3-L5	7525
T1-L6	9837			T3-L6	44485
				T3-L7	11341
				T3-L8	2518
				T3-L9	1951

Table 5.1: Breakdown of task label distribution. Task labels are listed as T#-L#, where T1-L1 represents Label 1 for Task-1.

Name	**#Layers**	**#Params**	**Size**
XLM-R(XLM-R Base)	12	85M	1.12GB
XtremeDistil	6	22M	91MB
MiniLM-L12(mMiniLMv2)	12	41M	236MB
MiniLM-L6(mMiniLMv2)	6	22M	215MB

Table 5.2: Description of model architectures evaluated. #Params refers to the number of transformer parameters.

datasets that covered 11 languages and evaluated how well the models perform in two settings: with a distillation teacher and without a teacher. The subset contained 57k responses labeled for Task-1 and 20k labeled for Task-2 and Task-3.

This experiment, as shown in Table 5.3, indicated that MiniLM (and its variants) would be easier to train and perform model distillation in our setting. Due to the above results, we targeted the MiniLM-L12 architecture for compressed models. Our definition of performing better, worse, or the same was based on the 95th percentile confidence interval of a random sample of 5 models trained from different random seeds. If we observe differences greater than these intervals we consider them significant; otherwise, we consider the result to be the

Method	**Task-1**	**Task-2**	**Task-3**
XLM-R	83.32	80.81	39.41
Xtremedistil (no teacher)	67.69	69.24	31.23
Xtremedistil (with teacher)	67.82	70.99	28.93
MiniLM-L12 (no teacher)	80.79	77.55	35.99
MiniLM-L12 (with teacher)	**81.43**	**78.44**	**36.54**

Table 5.3: Results on each task for each model architecture, reported in Macro-F1. All models were trained for two epochs, and the reported results are the per-task macro F1 scores.

same.

5.2.10 Cross-Lingual Model Results

Our goal in developing a cross-lingual model is to reduce the overhead of hosting multiple monolingual models. However, the single cross-lingual model should perform at least on par with the monolingual model. To test this assumption, we trained a single cross-lingual model and tested it across all languages. We then trained 12 separate monolingual models, starting from the publicly available XLMR-base pretrained weights (to avoid confounding factors from alternative monolingual base models). We then evaluated these monolingual models against the same cross-lingual evaluation dataset as a benchmark. A summary of results is shown in Table 5.4, where we report results for *fr*, *en*, *de*, and *ja* languages. We also evaluated 8 other languages and observed the same overall relative results. The best monolingual Task-1 result overall was 73.39 (*en*) and worst was 14.52 (*pl*), the corresponding cross-lingual reaching 79.12 (*pl*). The best cross-lingual result was 91.65 (*en*) and the worst was 71.05 (*ko*) with the corresponding monolingual result dipping to 48.84 (*ko*). We observe in every language we examined the cross-lingual model does better than the monolingual model, strongly supporting a move to cross-lingual modeling for our tasks.

5.2.11 Cross-Lingual Multi-Task (XLMT) Model Results

We are also interested in combining multiple tasks into a single model to reduce the engineering overhead of hosting our model. To evaluate whether our model maintained a similar

Train Lang	Eval Lang	Task-1	Task-2	Task-3
all	fr	**87.69**	**82.43**	**36.16**
fr		68.91	74.04	26.24
all	en	**91.65**	**79.67**	**40.47**
en		73.39	77.2	33.69
all	de	**86.07**	**77.74**	**34.71**
de		68.71	70.65	23.52
all	ja	**80.3**	**70.71**	**32.2**
ja		56.22	64.21	15.9

Table 5.4: Cross-lingual model comparison with monolingual models, evaluated on the same target language. Across all languages and tasks we evaluated, we observed the cross-lingual models to outperform monolingual models.

performance to single-task models, we evaluated the combined XLMT model compared to the single-task models for both XLM-R and mBert pretrained models. The experimental results in Table 5.5 show that the XLMT model performed similarly to, if not better, the single-task model on Task-1 and Task-2 prediction. For Task-3, we observed some significant degradation in the task performance.

To further confirm these results, we performed a similar analysis using another multilingual model, mBert. Using mBert, we again observed some modest gains for the first two tasks and significant degradation for the third task.

These results indicate our current multi-task architecture does benefit two of the three tasks. However, for the final deployment, it will be important to consider moving our third task into a separate model or develop alternative multi-task architectures to reduce the performance gap.

5.2.12 Compressed XLMT Model Results

In developing the XLMT model, engineering overhead was reduced from $16 \times 1 + 12 \times 2 = 40$ individual models to a single cross-lingual multi-task model or two based on the outcomes of Task-3 above. However, given the size of the XLM-Roberta model, the hosting costs associated with serving inference, specifically given the need for GPU instances to generate predictions at low latency, remained high. We focused on compressing the model itself to reduce this base cost and the latency of this model. As mentioned earlier, this compressing of the model, reducing its overall capacity, is in tension with the goals of maintaining the performance of the combined XLMR model. Our results in Table 5.6 show that simply performing structured layer pruning on the model resulted in some degradation of task performance. For

Model	Train Method	Eval Task	F1
XLM-R	multi-task	Task-1	82.23
		Task-2	76.03
		Task-3	38.32
	single-task	Task-1	81.12
		Task-2	74.67
		Task-3	51.27
mBert	multi-task	Task-1	78.88
		Task-2	75.27
		Task-3	35.88
	single-task	Task-1	78.63
		Task-2	74.31
		Task-3	51.12

Table 5.5: Task-specific results for cross-lingual single-task models and multi-task models. Macro-F1 results are reported on the full evaluation set, consisting of all languages (16 for Task-1, 12 for Task-2/3).

Task-1 with MiniLM-12 architecture, the larger the, the smaller architectures considered, we see about 1.6% relative degradation. MiniLM-6 shows 3.9% degradation, while XtremeDistil shows over 20% degradation. This same pattern holds for Task-2, and for Task-3, we see even less degradation for MiniLM-12. These results strongly favor MiniLM-12 and MiniLM-6 for compressing our specific use case.

Model	Task-1	Task-2	Task-3
XLM-R	82.23	76.80	35.90
MiniLM-L12	80.85	75.86	35.09
MiniLM-L6	78.97	72.42	35.34
XtremeDistil	61.83	61.59	24.00

Table 5.6: Results comparing the original MiniLM and XtremeDistil models with the full-size XLM-R model across Task-1, Task-2, and Task-3 macro-F1 scores.

5.2.13 Distilled XLMT Model Results

To address the degradation resulting from structured layer pruning, we incorporated some model distillation using the final layers of the full-size and compressed models. We explored using single multi-task teachers, task-specific teachers, single cross-lingual teachers, and language-specific teachers. However, we ultimately use cross-lingual task-specific teachers because the performance of Task-3 as a single task model outperformed the multi-task model, as shown in Table 5.5, and cross-lingual models consistently out-performed language-specific models as shown in Table 5.4. To provide additional model compression after enabling distillation, we trained the model with QAT to further reduce model complexity. To evaluate model speedup, each model was run for sequences of length 128 with batch size 32 on 1 Nvidia T4 GPU leveraging TensorRT [36]. Speedup was measured relative to the baseline model, XLM-R (fp32).

Model		**Speedup**	**Task-1**	**Task-2**	**Task-3**
XLM-R	(fp32)	x1	82.23	76.80	35.90
XLM-R	(int8 quantized)	x3.64	81.09	73.60	35.80
MiniLM-L12	(fp32)	x3.29	79.42	75.1	35.36
MiniLM-L12	(int8 quantized)	x8.11	79.29	73.69	35.71
MiniLM-L6	(int8 quantized)	x15.61	79.05	73.90	35.84
MiniLM-L12-mBert	(int8 quantized)	x8.11	79.05	73.48	35.57

Table 5.7: Results on model distillation and quantization aware training. Task-1, Task-2, and Task-3 results are reported in macro-F1 scores. XLM-R models were used as the teacher in all results, except for MiniLM-L12-mBert, which used mBert teachers.

The two best models after distillation were the quantized MiniLM-L6 model with 2.60% average relative task degradation and non-quantized MiniLM-L12 with 2.37% average relative task degradation. We found that quantized MiniLM-L6 was able to improve more from distillation than MiniLM-L12. While we are still investigating the cause, our current hypothesis is that the smaller model provides some regularization against overfitting versus the larger model. In terms of speedup, our quantized MiniLM-L6 model provided the most speedup at 15.61x speedup over the baseline. In our final assessment, we found that using task-specific model distillation on the MiniLM-L6 model with quantization provided a strong result in model size while maintaining model performance, as shown in Table 5.7. However, in considering the best model overall, the MiniLM-L12 in Table 5.6 provided a minor overall degradation of 1.71% and a modest speedup of 3.29x.

5.2.14 Business Impact

An implementation of these compression techniques and choices was developed in our production training system. At Qualtrics, this system has generated significant business im-

[36]`https://developer.nvidia.com/tensorrt`

pact across customers, new features, financial, environmental, and operational costs, system flexibility, and robustness.

5.2.15 Feature impact

Given the compressed and multi-task models' speedup, we observe significant increases in throughput across use cases. This enables us to serve more customers and help new features for previously too compute-intensive customers. For example, this speedup enables ML pipelines that run 3-4 models serially in the same time window as a single model. Additionally, the flexibility of cross-lingual models enables us to serve more customers in more languages without large training sets or comprehensive language specialization.

5.2.16 Financial impact

Conservatively, we estimate approximately 44% savings in terms of hardware cost from the developments in compression of the multi-task cross-lingual model compared to an uncompressed system at similar latency and throughput. In addition, by combining multiple tasks invoked with a similar load into a single model, we achieve a fraction of the total inference cost.

These savings are driven by several factors: reducing base instance capacity, reducing the amount of dynamic scaling, and allowing for the deployment of lower-cost hardware. We note that the savings are limited by the ongoing cost of the base capacity, even when reduced, which creates a floor for cost savings even with multiple times faster models.

Currently, we deploy our models on a public cloud with a cost of approximately \$80K/-month/model. The compression technique used in this paper reduces cost by 44%, resulting in \$35K savings/month for a single model compared to the current model in production for the same tasks. As we push our compression framework to various other NLP systems currently developed by this group, it can result in potential yearly cost savings in the single-digit millions of dollars. Considering current macroeconomic conditions, when there is an industry-wide need for financial cost reduction, significant savings can strengthen the fundamentals of a typical SaaS company like Qualtrics.

5.2.17 Ethical impact

We are pleased that our efforts to compress models will support environmental sustainability. By reducing the amount of power and resource needed to run the same inference, we anticipate a meaningful impact on the environmental footprint. However, we are unable to quantify it concretely at this time.

5.2.18 Operational Impact and Robustness

While MiniLM-L6 provided better speedup, business needs required the lower degradation provided by MiniLM-L12 for the first set of models. By leveraging the compressed XLMT model, we enable additional flexibility in production deployment scenarios, including different instance types (CPU, GPU, custom accelerators). This allows us to serve high throughput inference while balancing cost. This was not previously viable with larger models, which required GPUs to serve high throughput inference. By enabling this flexibility, we also improve system robustness, as the models are robust to instance unavailability or instance downtime for any instance.

Additionally, savings from moving to a single multi-task model results in a reduced workload for our engineering teams, removing per-task deployments and deployment pipelines and lowering the barriers to new tasks and language support. Specifically, multi-task and cross-lingual modeling reduces the number of models for these tasks from 36 potential models (12 languages, three tasks) to a single model, reducing operational costs from 6-7 on-call/operations engineers to 1. Compression reduces this cost by lowering latency and increasing throughput, reducing the operational cost of mitigating latency spikes and scaling issues.

5.2.19 Conclusion

We have presented a case study into the Qualtrics approach for leveraging cross-lingual and multi-task modeling techniques in combination with model distillation and quantization techniques to develop models that can handle traffic volumes at scale. The results show that these methods can be combined effectively to reduce deployment overhead and maintenance and achieve up to 15.61x speedup with 2.60% average degradation for our multi-class classification tasks. Our results explore boundary cases where compression works well and can degrade past business requirements; combining up to 12 languages works well; combining tasks works well, and where it does not. These approaches have been necessary for us to scale up our unique text classification problems in the growing field of experience management. We anticipate these results will help guide other groups hoping to reduce model inference costs and contribute to future theoretical work around the trade-off between model compression and model consolidation. Looking forward, we plan to apply these methods to more complex sequence labeling tasks and explore additional techniques such as model sparsity and neural architecture search to see if even faster models can be developed with acceptable levels of model performance.

5.3 TO ASYMMETRY AND BEYOND: STRUCTURED PRUNING OF SEQUENCE TO SEQUENCE MODELS FOR IMPROVED INFERENCE EFFICIENCY

5.3.1 Overview

Sequence-to-sequence language models can be used to produce abstractive summaries which are coherent, relevant, and concise. Still, model sizes can make deployment in latency-sensitive or web-scale implementations difficult. This section studies the relationship between model size, structured pruning, inference efficiency, and summarization accuracy on widely used summarization datasets. We show that model accuracy is tied to the encoder size while inference efficiency is connected to the decoder. Using asymmetric pruning can lead to nearly 3x improvement in inference latency with 1 point loss in Rouge-2. Moreover, we find both the average degradation and the role of asymmetry to be consistent across model sizes and variations in datasets. We release our code[37], training regimes, and associated model [38] for broad usage to encourage usage and experimentation.

5.3.2 Introduction

The application of sequence-to-sequence language models has become an important tool for natural language processing tasks such as machine translation [209], audio transcription [38], and abstractive summarization [11]. Sequence-to-sequence models effectively turn each of these aforementioned tasks into two-step problems: extraction and generation, and heavily condition the generation on the input.

Besides ensuring on-topic responses sequence to sequence models have the added benefit of being able to map inputs to targets with varying lengths and modalities in ways encoder or decoder-only systems cannot.

When used for abstractive summarization, sequence-to-sequence modeling has two steps, extraction using the encoder and generation using the decoder, which usually involves repeated execution until an end-of-sequence token is emitted. While the cost of encoder execution is essentially fixed on the batch size, the cost of decoder execution can be highly variable

[37]https://github.com/spacemanidol/Efficient-Web-Scale-Absractive-Summarization

[38]https://huggingface.co/spacemanidol

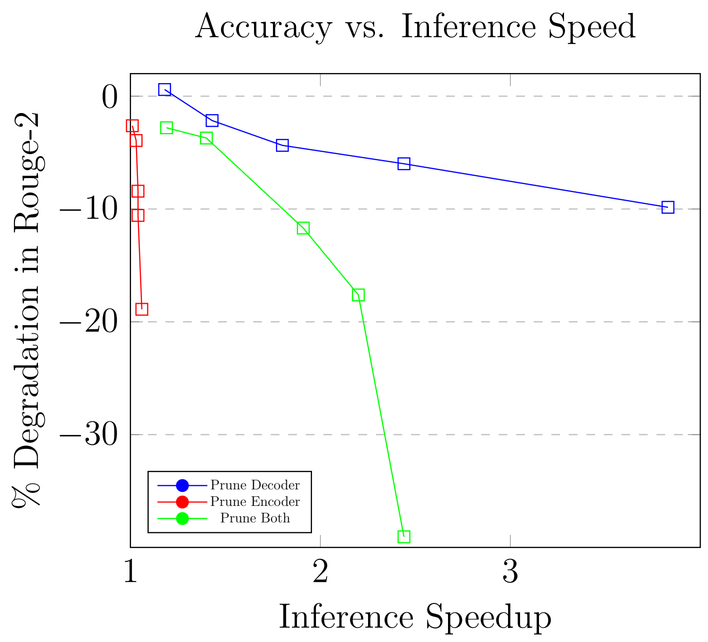

Figure 5.2: Impact of Asymmetrical Pruning on inference speedups and ROUGE-2 degradation on Query Independent Web Summarization. Inference Time is the mean inference time for a batch size of 1 on an A10 GPU over seven iterations.

and difficult to predict. Despite the broad study of sequence-to-sequence models and how they compress research which studies the role of model symmetry as applied to inference efficiency and model accuracy is lacking.

Recent advances in scaling language models have led to a wide study on *scaling laws* as applied to language model performance [22], training data size [102], machine translation [210], and even reinforcement learning [211].

We build on this work and study the impact of scaling on abstractive summarization and what role model asymmetry has in it.

This asymmetry can manifest in various ways, such as the number of layers and hidden units in the encoder and decoder and the type of attention mechanisms used.

In this section, we explore the role of asymmetry in the number of layers in encoder-decoder language modeling for summarization and its impact on the performance of these models. As shown in figure 5.2, the symmetry of pruning drives the impact on accuracy and inference speedups for sequence-to-sequence models. Pruning the encoder portion of the network leads to virtually no improvement in inference speed at the expense of accuracy. Pruning the decoder provides speedup with minor losses in accuracy. The following research questions drive our work:

- What scaling laws can be observed in abstractive summarization?
- What impact does encoder-decoder asymmetry have on abstractive summarization

accuracy?

- What impact does encoder-decoder asymmetry have on abstractive summarization inference efficiency?
- What is asymmetries impact on accuracy and inference efficiency does scale have in encoder-decoder models for abstractive summarization?

It is in answering these questions that we deliver the following contributions:

- We present the first robust study on scaling laws applied to the compression of sequence-to-sequence modeling.
- We demonstrate that the asymmetric inference cost of sequence-to-sequence models leads to asymmetric pruning for optimal inference efficient compression.
- We empirically demonstrate on a wide variety of benchmarks how Asymmetric Compression can lead to a 2.7x inference speedup with no loss in accuracy on the XSUM dataset.

5.3.3 Scale and Abstractive Summarization

Sequence-to-sequence language models such as BART [212], T5 [11], and PEGASUS [37] combine transformer encoders and decoders to produce models which can adapt to novel tasks and reach top performance on tasks ranging from information retrieval [213] to summarization [11].
We focus on the instruction-tuned FLAN-T5 models [214] as their performance is competitive and they feature wide variations in model size ranging from 60 million to 11 billion parameters and given the cost of training the larger variants, focus on the small, base, and large variants.
Abstractive summarization is a method of sequence compression where a source document D is transformed into a target document d_{sum}, which is shorter but faithful to the input.
Using a sequence-to-sequence model such as T5 [11], a transformer-based encoder (enc) produces a contextual document representation e, and a transformer-based decoder generates d_{sum} one token at a time conditioned on e. Models are initially pre-trained in a self-supervised fashion and then fine-tuned using a set of source and target documents where each document $d \in D$ contains a d_{sum}, which is both shorter than d and true to the source.
Datasets of use are a combination of public and academic benchmarks and a proprietary

web search dataset. The CNN/DailyMail (CNNDM) [215] and XSUM [40] datasets are based on the summarization of English new language models. The CNN/Dailymail abstractive text summarization dataset is a collection of documents from CNN and Daily Mail annotated for descriptive summaries. The dataset consists of 311,672 documents and their respective summaries.
XSum is a dataset for training and evaluation of single-source document summarization. Given a document X the goal is to produce a summary Y which answers the question *What is a document about?*. It is comprised of 226,711 articles along and their human-generated summary. The documents come from BBC articles covering broad topics like news, sports, and technology from 2010-2017.
The Query Independent Web Summary (QIWS) is a proprietary corpus of abstractive summaries of web pages that are used to create informative contextual snippets for search engine users. It is important to note the differences in compression factor in each dataset as each impact how decoder-driven inference latency is. Further information on the makeup of each dataset can be found in table C.1

Metrics For each dataset, we evaluate model performance by measuring the ROUGE-

Model	R-2	Gain	BS 1 Latency	Impact	BS 8 Latency	Impact	BS 16 Latency	Impact
small	17.55	0.00%	138	1	230	1	330	1
base	19.77	12.63%	199	1.44	550	2.39	931	2.82
large	21.15	20.51%	445	3.22	1480	6.43	2700	8.18

Table 5.8: Impact of scale on inference throughput for abstractive summarization models trained on the XSUM dataset. Latency is measured in MS/batch, and the impact is the impact to latency vs. the small model

Model	R-2	Gain	BS 1 Latency	Impact	BS 8 Latency	Impact	BS 16 Latency	Impact
small	29.03	0	524	1	653	1	729	1
base	34.19	17.77%	746	1.42	1060	1.62	1310	1.80
large	37.37	28.72%	1,430	2.73	2240	3.43	3320	4.55

Table 5.9: Impact of scale on inference throughput for abstractive summarization models trained on the QIWS dataset. Latency is measured in MS/batch, and the impact is the impact to latency vs. the small model

1 (R-1), ROUGE-2 (R-2), ROUGE-L (R-L), RougeSum-L (RSL) [39] [216], and Generation Length (GenL) on the test portion of the dataset. To aid the reproducibility and extension of our work, we experiment using HuggingFace's Transformers [40], release our training and

[39]Rouge-L is sentence level vs. RougeSum-L is summary level
[40]https://github.com/huggingface/transformers

Model	R-2	Gain	BS 1 Latency	Impact	BS 8 Latency	Impact	BS 16 Latency	Impact
small	11.09	0	171	1.00	252	1.00	344	1.00
base	15.69	41.50%	255	1.49	550	2.18	845	2.46
large	16.34	47.41%	525	3.07	1370	5.44	2300	6.69

Table 5.10: Impact of scale on inference throughput for abstractive summarization models trained on the CNNDM dataset. Latency is measured in MS/batch, and the impact is the impact to latency vs. the small model

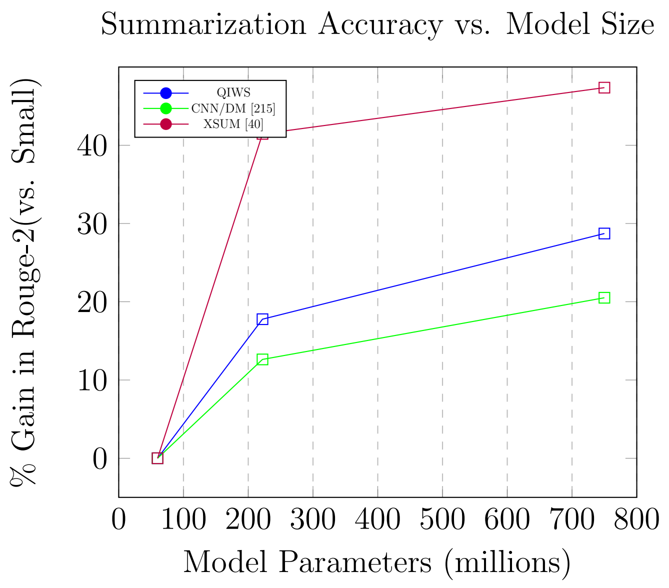

Figure 5.3: Model Size vs. Gain to summarization accuracy as measured by the relative Gain in rouge-2 vs. the small model.

pruning scripts [41] and model variants for datasets that are publicly available datasets [42].

5.3.4 Scaling Laws for Abstract Summarization

To study the role of scale in abstractive summarization, we train small, base, and large models of the three datasets mentioned above. We do not study the XL (3B) and XXL (11B) as they are expensive and slow to train.

For all of our experiments, we train on various hardware but fix the batch size to 64 using gradient accumulation and leverage the hyperparameters in C.2. While further hyperparameter optimization and instruction tuning would likely lead to further gains in accuracy, our work is not focused on absolute Gains but on the relative relation of scale.

[41]https://github.com/spacemanidol/Efficient-Web-Scale-Absractive-Summarization
[42]https://huggingface.co/spacemanidol

As shown in 5.3, C.3, C.4, and C.5, there is a substantial role between scale and performance, but there is a substantial variation across datasets.
Datasets with short candidate summaries, such as XSUM, see nearly three times the impact compared to the long summaries of QIWS and XSUM. During qualitative evaluations, the role of scale can easily be observed as smaller models generate more short keyword summaries while introducing scale makes responses more natural.

		Small		Base		Large	
l_{enc}	l_{dec}	R-2	R	R-2	R	R-2	R
6	6	29.03	100.00%	34.19	100.00%	37.37	100.00%
6	5	28.90	99.55%	34.00	99.44%	37.59	100.59%
6	4	28.56	98.40%	34.50	100.91%	36.56	97.84%
6	3	27.94	96.24%	33.70	98.58%	35.74	95.64%
6	2	24.85	85.61%	31.93	93.38%	35.13	94.01%
6	1	15.41	53.08%	28.05	82.03%	33.69	90.15%
5	6	27.92	96.17%	33.57	98.18%	36.39	97.38%
4	6	27.75	95.60%	33.06	96.69%	35.90	96.07%
3	6	25.20	86.82%	32.23	94.28%	34.22	91.58%
2	6	23.67	81.55%	27.47	80.35%	33.42	89.43%
1	6	18.23	62.79%	25.57	74.78%	30.31	81.11%
5	5	26.82	92.38%	32.88	96.18%	36.32	97.20%
4	4	26.62	91.72%	32.81	95.96%	35.98	96.29%
3	3	23.12	79.64%	28.70	83.95%	33.00	88.31%
2	2	19.14	65.92%	26.53	77.60%	30.78	82.38%
1	1	6.09	20.99%	19.64	57.43%	22.77	60.94%

Table 5.11: Relation between scale and asymmetry on model performance on the QIWS dataset

5.3.5 Inference Benchmark

To evaluate the impact of asymmetry on inference, we run experiments on the throughput of each model. Using an A10 GPU and the models from our QIWS datasets, we evaluate performance with a max sequence length of 1024, a max summary of 256, and batch sizes 1, 8, and 16 using native inference in PyTorch. We report the mean and standard deviation of timings on seven runs.
In comparing the impact of scale on R-2 vs. the effects on latency across batch sizes in 5.8, 5.10, 5.9 it becomes clear that larger models are more expensive to execute significantly as batch sizes increase. This is because of potential differences in output length within a batch as the batch completes when all sequences have produced an *EOS* token. To alleviate this issue bottleneck, improved streaming methods for improved batching have been proposed

[217] but can be challenging to manage.

5.3.6 To Asymmetry and Beyond

While prior work has studied how to improve inference and tangentially explored the asymmetry between the encoder and decoder, we study that explicitly and across model scales. We focus our studies on **structural pruning** as inference gains are easy to realize, and this approach is highly compatible with other methods like quantization and unstructured pruning.

Following Shleifer et al., we adopt the **S**hink and **t**hen **f**ine (STF) tune approach for compression. First, a model is trained until convergence on a fine-tuning summarization task. Then, entire layers are removed from the encoder, decoder, or both, and the model is further fine-tuned until it has re-converged.

Each model we study has a uniform number of encoder and decoder layers, so we prune only the encoders, decoders, and a symmetric combination of the two combinations. We used our three scales of uncompressed models (small, base, large), and we pruned the model in multiples of 1 on the encoder, the decoder, and both. After pruning, models are fine-tuned again and evaluated. This means that for each dataset, we have 16 variants for each model size leading to 48 models per dataset and 144 models overall.

Given the wide number of models and the cost of multiple seeds or model-specific optimization, we train each model once and do not optimize the parameters for each model. While this leads to a worse-than-ideal performance, our goal is not to hyper-optimize models but explore where there is high sensitivity. To save space, we use the shorthand l_{enc} and l_{dec} to refer to the number portion of transformer encoder and decoder layers (out of 6), and R refers to the percentage performance recall vs. uncompressed baseline. Detailed results have been moved to the C.1.3 to save space.

5.3.7 Scale and Pruning

Looking at abridged results in 5.11, 5.12, and 5.13, there is a clear scaling law as smaller models see much larger drops in performance when compressed to the same degree. For example, on the QIWS dataset, compression to $\frac{1}{6}$ of the layers on the encoder and decoder cause an 80% drop in R-2 on a small model but only 40% on the larger model. This scale comparison is 65% to 26% on CNNDM and 64% to 45% on XSUM datasets.

Similar scaling results hold with encoder or decoder pruning, where compressing large mod-

		Small		Base		Large	
l_{enc}	l_{dec}	R-2	R	R-2	R	R-2	R
6	6	17.55	100.00%	19.77	100.00%	21.15	100.00%
6	5	17.68	100.74%	19.92	100.76%	21.30	100.69%
6	4	17.27	98.36%	19.85	100.42%	21.32	100.81%
6	3	16.40	93.43%	18.85	95.37%	21.08	99.66%
6	2	15.35	87.42%	18.68	94.51%	20.67	97.73%
6	1	11.33	64.57%	16.48	83.38%	19.49	92.12%
5	6	17.69	100.81%	19.92	100.76%	21.13	99.88%
4	6	17.35	98.84%	19.67	99.50%	20.83	98.47%
3	6	16.80	95.70%	18.85	95.37%	20.53	97.06%
2	6	15.54	88.51%	18.22	92.14%	19.74	93.33%
1	6	13.31	75.83%	17.06	86.27%	18.68	88.31%
5	5	17.07	97.23%	19.72	99.74%	21.23	100.34%
4	4	16.20	92.28%	19.17	96.99%	20.90	98.81%
3	3	14.91	84.95%	17.46	88.29%	20.13	95.16%
2	2	11.97	68.17%	15.87	80.26%	18.47	87.30%
1	1	6.05	34.45%	12.23	61.88%	15.51	73.32%

Table 5.12: Relation between scale and asymmetry on model performance on the CNNDM dataset

		Small		Base		Large	
l_{enc}	l_{dec}	R-2	R	R-2	R	R-2	R
6	6	11.09	100.00%	15.69	100.00%	16.34	100.00%
6	5	11.61	104.74%	15.27	97.35%	19.80	121.16%
6	4	11.43	103.12%	14.91	95.03%	19.30	118.09%
6	3	11.24	101.36%	15.40	98.17%	18.92	115.77%
6	2	10.53	94.98%	15.19	96.82%	17.96	109.93%
6	1	6.03	54.42%	13.73	87.53%	16.47	100.76%
5	6	11.18	100.82%	15.92	101.47%	19.43	118.88%
4	6	10.61	95.68%	14.10	89.91%	18.33	112.16%
3	6	10.11	91.16%	13.84	88.21%	16.90	103.39%
2	6	8.59	77.52%	12.10	77.12%	14.97	91.61%
1	6	7.70	69.43%	10.27	65.47%	12.52	76.63%
5	5	10.73	96.76%	15.72	100.22%	19.18	117.38%
4	4	10.19	91.96%	14.30	91.15%	17.56	107.43%
3	3	9.50	85.69%	12.44	79.32%	15.89	97.21%
2	2	7.31	65.91%	10.67	68.05%	12.15	74.34%
1	1	4.00	36.09%	7.74	49.35%	8.96	54.86%

Table 5.13: Scale and Pruning on XSUM dataset

els lead to a 5x lower loss in performance than small models. As the model's size grows, the impact of decoder vs. encoder-only pruning becomes more muted. On the CNNDM dataset, the gap between the decoder only and encoder only pruned to $\frac{1}{6}$ is 10% with the FLAN-T5 small but only 4% with the large variant. When comparing asymmetric and symmetric, the

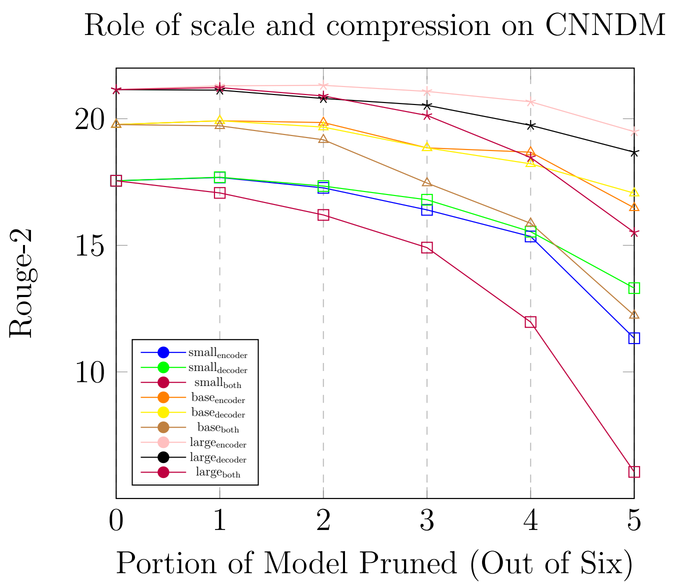

Figure 5.4: Relationship between model compression, model size, and summarization accuracy measured by rouge-2 vs. Number Layers. small$_{encoder}$ refers to a FLAN-T5 small which has only pruned the encoder, small$_{decoder}$ for only the decoder, and small$_{both}$ for the encoder and decoder

gap is even further pronounced where the small gap is 30% while the large is 20%.
As shown in figure 5.4, the impact of compression becomes more muted as the model size grows. In other words, larger models are more compressible and amenable to asymmetry in this compression.
The impact of asymmetry is easiest to understand as it is not surprising that complete pruning of a model leads to higher losses than partial pruning across datasets and model sizes. While this finding is not immediately surprising, evaluating the inference costs becomes important.

5.3.8 Inference Benchmarks

We evaluate the impact of asymmetry in a similar method to our scale experiments. Using an A10 GPU, we evaluate performance for summarization on a portion of each model's respective evaluation datasets with a max sequence length of 1024, a max summary length of 256, and batch sizes 1, 8, and 16.

Looking at the focused set of results for large models across datasets in table 5.14 on the impact of R-2 vs. inference speedup, we can see a clear relationship between asymmetry

		QIWS		CNN/DailyMail		XSUM	
l_{enc}	l_{dec}	Impact	Speedup	Impact	Speedup	Impact	Speedup
6	3	-4.36%	1.80	-0.34%	1.65	15.77%	1.64
6	2	-5.99%	2.44	-2.27%	2.03	9.93%	2.07
6	1	-9.85%	3.83	-7.88%	2.70	0.76%	2.71
3	6	-8.42%	1.04	-2.94%	1.14	3.39%	1.16
2	6	-10.57%	1.04	-6.67%	1.19	-8.39%	1.21
1	6	-18.89%	1.06	-11.69%	1.27	-23.37%	1.30
3	3	-11.69%	1.91	-4.84%	1.94	-2.79%	2.06
2	2	-17.62%	2.20	-12.70%	2.78	-25.66%	2.83
1	1	-39.06%	2.44	-26.68%	4.96	-45.14%	4.84

Table 5.14: Relationship between accuracy and speedup of encoder only, the decoder only, encoder and decoder pruning on FLAN-T5 Large models on CNN/DM, XSUM, and QIWS. Speedup is measured by comparing the improvements in latency for batch size one vs. the uncompressed baseline. The impact is the relative loss of Rouge-2 of compressed models vs. the uncompressed baseline.

and inference efficiency. While detailed inference results can be found in the appendix C.1.4 on this focused set of results, we can see that pruning only the encoder leads to no more than 30% improvement in inference efficiency at a sizable loss in accuracy. Pruning the model symmetrically leads to realizable inference improvements of up to 5x at the expense of summarization accuracy.

Alternatively, when only the decoder is pruned, it is possible to see most of the inference speedups seen during constant pruning with a substantially lower impact on accuracy. On the CNN/DM dataset, constant pruning leads to 8% better inference but losses nearly four times the performance of non-uniform compression.

		Small		Base		Large	
l_{enc}	l_{dec}	Impact	Speedup	Impact	Speedup	Impact	Speedup
6	6	-3.76%	1.79	-1.42%	1.76	-4.36%	1.80
6	6	-14.39%	2.69	-6.62%	2.13	-5.99%	2.44
6	6	-46.92%	3.97	-17.97%	3.69	-9.85%	3.83
3	3	-13.18%	1.02	-5.72%	1.04	-8.42%	1.04
2	2	-18.45%	1.02	-19.65%	1.05	-10.57%	1.04
1	1	-37.21%	1.03	-25.22%	1.06	-18.89%	1.06
3	3	-20.36%	1.40	-16.05%	1.86	-11.69%	1.91
2	2	-34.08%	1.30	-22.40%	2.48	-17.62%	2.20
1	1	-79.01%	3.91	-42.57%	3.95	-39.06%	2.44

Table 5.15: Relationship between accuracy and speedup of encoder only, decoder only, encoder and decoder pruning on FLAN-T5 models on QIWS concerning model size. Speedup is measured by comparing the improvements in latency for batch size one vs. the uncompressed baseline. The impact is the relative loss of Rouge-2 of compressed models vs. the uncompressed baseline.

l_{enc}	l_{dec}	Impact	Speedup (BS1)	Speedup (BS8)	Speedup (BS16)
6	3	-0.34%	1.65	1.18	1.15
6	2	-2.27%	2.03	1.25	1.22
6	1	-7.88%	2.70	1.36	1.29
6	3	-2.94%	1.14	1.48	1.54
6	2	-6.67%	1.19	1.68	1.89
6	1	-11.69%	1.27	2.21	2.43
3	3	-4.84%	1.94	1.96	1.97
2	2	-12.70%	2.78	2.88	2.92
1	1	-26.68%	4.96	5.54	5.64

Table 5.16: Relationship between accuracy and speedup of encoder only, decoder only, encoder and decoder pruning on FLAN-T5 large models on CNN with variation in inference batch size. Speedup is measured by comparing the improvements in latency vs. the uncompressed baseline at various batch sizes. The impact is the relative loss of Rouge-2 of compressed models vs. the uncompressed baseline.

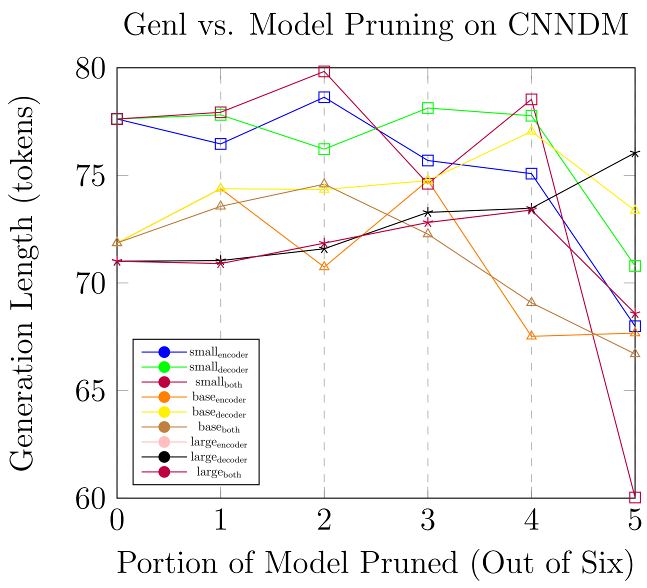

Figure 5.5: Role of scale and compression on generation length

5.4 DISCUSSION

5.4.1 Scale, Inference, and Pruning

As shown in table 5.15, the gains found by pruning are extremely consistent independently with scaling. Pruning only the encoder leads to a 4-6% improvement in latency, and pruning just the decoder leads to 400%, as does uniform compression. This is expected as structural pruning removes a constant portion of the network, which leads to consistent latency gains irrespective of model scale.

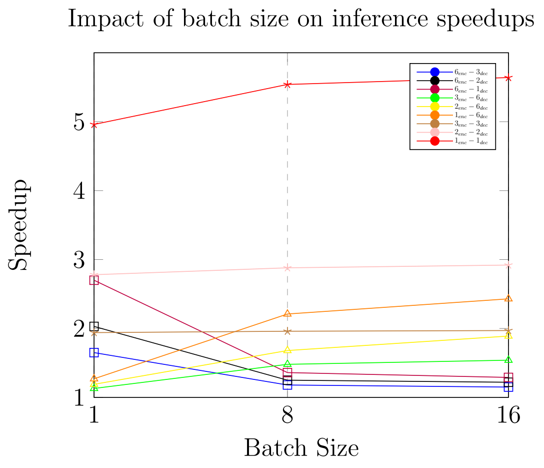

Figure 5.6: Relationship between inference batch size and realized inference speedup with uniform and no uniform pruning of FLAN-T5 large on CNNDM

5.4.2 Scale, Pruning and Generated Length

Despite expecting a significant trend in the role of scale and pruning in a generation, we do not see any noticeable trends. As shown in figures 5.7 and 5.5, there is no discernible trend of the Role of scale and pruning in generation length. There is a minor jump in generation length from FLAN-T5 small to FLAN-T5 base across all datasets but no such jump from FLAN-T5 base to FLAN-T5 large. We believe this is because the smaller models are less fluent and need more tokens to ensure accurate coverage. As models scale, this is no longer needed, and the models converge to a uniform summary length.

5.4.3 Asymmetry Meet Large Batches

Despite the allures of asymmetrical pruning, it is not without fault. As shown in table 5.16 and figure 5.6, the improvements in inference efficiency are heavily influenced by the batch size. When the batch size is minimal, the difference in the type of non-uniformity has a significant impact on inference efficiency. As batches scale, the speedup from encoder only or decoder only becomes much closer and becomes minor when compared to uniform methods. This indicates why further work on improving generative inference methods is highly relevant, as this problem impacts other efficiency-driven processes like CALM [113].

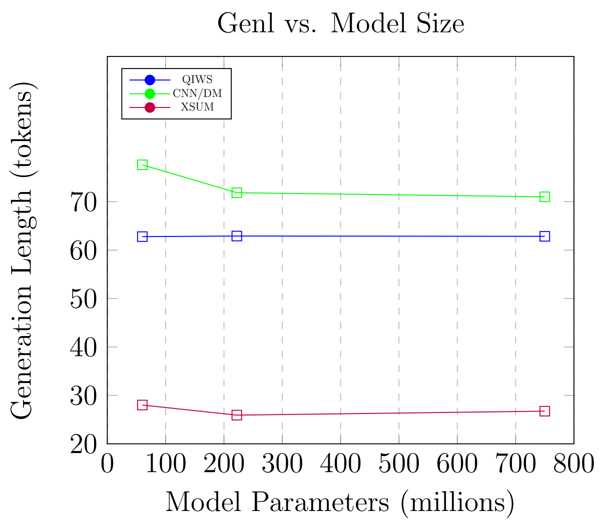

Figure 5.7: Role of scale on generation length

5.4.4 Conclusion and Future Work

In this work, we explore the role of symmetry in the pruning of sequence-to-sequence models for abstractive summarization, finding that pruning asymmetrically can lead to inference speedups with low losses in accuracy. Our work also explores the relationship between model scale and the sensitivity to pruning, finding that larger models see lower losses when pruned. This compresses FLAN-T5 models to deliver 3x inference gains with a 1 Rouge-2 point loss.

In future work, we seek to study how pseudo labeling, early exiting, and quantization can be combined to improve further the inference efficiency of sequence-to-sequence models.

5.5 CONCLUSION AND KEY TAKEAWAYS

In conclusion, in this chapter, we study the effective use of pruning, quantization, and knowledge distillation to improve the inference efficiency of web-scale workloads. In the exploration of optimal compression mechanisms, we find the following key takeaways:

Larger models compress better. In multi-task multi-label compression, we find that larger models can be quantized with no loss in accuracy while the smaller models see higher losses. Moreover, smaller models require specialized compression regimes such as multi-teacher knowledge distillation. In abstractive summarization, larger models are significantly more robust to structural pruning. Small models see noticeable impacts on accuracy even with

minor compression, but larger models can be more heavily pruned before they see losses.
Exploiting the structure of inference is paramount. While training of large models favors uniformity, understanding the costs of inference is key to compression. Sequence-to-sequence models leverage the decoder substantially more than the encoder for inference. This leads to minor gains in inference efficiency with compression on the encoder and major gains on the decoder. Moreover, since sequence-to-sequence models deeply leverage the contextual representation extracted by the encoder, there is a higher impact on accuracy when the encoder is compressed vs. the decoder.
Scaling workloads requires trade-offs. Large web-scale inference workloads actively balance cost efficiency, model quality, maintainability, and expandability. As a result, large workloads are constantly trading off the impact of improvements on one portion with losses in another. While using advanced compression techniques can mitigate the impact of compression, it can introduce substantial overhead in the training regime.

CHAPTER 6: CONCLUSION

6.1 OVERVIEW

In the previous few chapters, our work has demonstrated how language models can be used for web-scale understanding, generation, and retrieval. This thesis's main contribution to natural language processing and web-scale information retrieval is a robust and thorough evaluation of methods and approaches for improving inference efficiency and retrieval accuracy. From the use of quantization in multi-task classification to structured pruning for text generation, compression can lead to major speedups with little to no impact on accuracy. In this chapter, we will discuss key overall takeaways supported by our research. Additionally, the chapter discusses the limitations of the research and potential directions for future research.

The practical implications of the research are also discussed, highlighting the potential applications of the findings in real-world scenarios. Finally, we summarize our main contributions and impact, emphasizing the significance of the research in advancing state-of-the-art natural language processing and web-scale information retrieval.

6.2 TRAIN LARGE THEN COMPRESS

With the growth of language model sizes, understanding scale's impact on compressibility is paramount. Understanding that larger models are more sample-efficient, questioning how to trade off the model's initial size and compression ability is natural.

First, as our work on asymmetric generation shows, larger models compress better. Independent of pruning approaches, as models grew from 60m parameters to over 700m, the impact of structured pruning on model performance dropped by nearly 4x. As shown in Chapter 2, we see a similar trend in our experimentation with unstructured pruning as 3-layer models lose 2-3x more performance (relative accuracy loss) than 12-layer models.

Given larger models compress better when attempting web-scale deployments, a practical approach is to train the largest model one can, followed by compression until it reaches the desired inference efficiency.

It is worth noting that the reason larger models compress better may be tied more closely to being under-optimized. Training larger models is significantly more complex and expensive, and as a result, small models are easier to optimize to a given corpus. Optimization in pretraining commonly requires 10-100x the pretraining corpora and training length but leads to large improvements in model accuracy, such as the gap between BERT and RoBERTa. However, increasing training regimes by such ratios is impractical as models scale. Moreover, given the costs of training at scale, larger models may be under-optimized. Since larger models are more under-optimized, these models are easier to compress as the architecture still has plasticity. As shown in the performance gap in compressing RoBERTa vs. BERT, more optimized models are harder to compress. The loss in accuracy on SQUAD using the same pruning method leads to a 5-6x loss in accuracy with the optimized model (RoBERTa) vs. the under-optimized model (BERT). Further study is needed to confirm this hypothesis.

6.3 WEB SCALE DEPLOYMENTS ARE ALL ABOUT TRADE-OFFS

In many deployments of language models, model accuracy and inference efficiency are at odds. Larger models commonly lead to improvements in accuracy at the cost of inference latency. While it is possible to compress models without losing accuracy, these approaches do not always lead to the needed improvements in inference efficiency. As a result, the breadth of a model deployment is mostly driven by how sensitive the workload is to losses in model sensitivity. Tasks where a 10% loss in model performance leads to minor variations in outcomes are attractive to intensive compression, while tasks that cannot may only slightly compress.

Moreover, trade-offs are involved in selecting the appropriate hardware and software in-

Layers	Baseline (FP32)	Cost	Quantized (Int8)	Cost
12	106	\$4.99	423	\$1.25
9	139	\$3.79	558	\$0.95
6	172	\$3.06	689	\$0.77
3	299	\$1.76	1198	\$0.44
2	441	\$1.20	1766	\$0.30
1	661	\$0.80	2643	\$0.20

Table 6.1: Estimated inference throughput and cost efficiency of compressing a Language Model with 12 encoder layers. Cost represents the number of dollars required to perform one million inferences. Inference costs and throughput are based on a g4dn.xlarge on AWS, which is \$0.5260/hr. Since the GPU in this variant does not support sparsity, we do not include performance numbers for sparse or sparse quantized models.

frastructure for deploying language models at a web scale. For example, using specialized hardware such as graphics processing units (GPUs) can significantly improve the perfor-

Layers	Baseline (FP32)	Cost	Quantized (Int8)	Cost	Sparsity	Cost	Sparsity + Quantized	Cost
12	47	\$7.65	189	\$1.91	156	\$2.33	591	\$0.61
9	63	\$5.72	253	\$1.43	208	\$1.74	791	\$0.46
6	90	\$4.00	362	\$1.00	297	\$1.22	1130	\$0.32
3	166	\$2.18	664	\$0.54	546	\$0.66	2075	\$0.17
2	230	\$1.57	919	\$0.39	756	\$0.48	2871	\$0.13
1	379	\$0.96	1514	\$0.24	1245	\$0.29	4732	\$0.08

Table 6.2: Estimated inference throughput and cost efficiency of compressing a Language Model with 12 encoder layers. Cost represents the number of dollars required to perform one million inferences. Inference costs and throughput are based on a m5.4xlarge on AWS which is \$0.768/hr

mance of language models but comes at the cost of increased infrastructure costs.
When deciding what size/scale of the model is needed for a given task, the most straightforward approach is to estimate the scale and budget needs. As shown in Table 6.2 and 6.1, a simple framework can be built around the cost of serving relative to compression approaches. These tables are built around a real-time classification system that processes 1 million text samples daily in a batch fashion. For this workload, using an uncompressed model on a CPU is still affordable, with a daily cost of \$7.65, which probably does not require further compression. Instead, consider a workload that needs to run a process on a much larger corpus, say a web index of 10,000,000,000 documents. Running that same uncompressed model would cost \$76,500 a day! For that same context using an undersized 2-layer, the sparse quantized model would be \$1,300 or nearly 60x cheaper. As shown in our experimentation with KALE in 4, going from 12 layers to 2 on the NQ dataset leads to only a 1.5% loss in retrieval accuracy, which is often acceptable for the 60x cost savings.

6.4 COMPRESSION APPROACHES ARE ADDITIVE AND ADAPTABLE

There are many effective ways of compressing models, and each can be applied at many points of model creation. As shown in chapter 3, compression approaches are mostly additive. If the impact on inference efficiency of unstructured pruning is 2x while quantization is 2x, then using both leads to 4x speedup. In practice, the realized gains of additivity are seen as some non-linearity; they can effectively be combined for maximal results.
Besides being addictive, various compression approaches have the benefit of being able to be applied at many stages of model creation. As shown in 3, compression can be used during pre-training or fine-tuning, and compressed models can be adapted to specialized domains without further optimization or reduction.

6.5 REPRESENTATION ALIGNMENT IS CRUCIAL

Compression can be thought of as removing portions of a model while striving to lose none of the model's expressivity. While it is possible to do so without further specialization, it cannot be easy. As a result, approaches that improve the compressed model's ability to emulate the original model are desirable. One of the most effective approaches is representation alignment, also called knowledge distillation.
As shown in chapter 5, 4, 3, the impact of representation alignment allows models to perform well when compressed and on robust inputs. This alignment can happen using the cosine distance, KL divergence of their hidden states, or even just training on pseudo labels.
In 4.3.2, we discuss how representation alignment can be used to make models more robust, and we take the exact formulation and use it in B.1.4 to compress the model.

6.6 LIMITATIONS

While our work has focused on showing how widely compression approaches can be deployed, compression can be difficult to implement and realize. Our experiments required a high degree of experimentation on the role of compression on model performance before we could effectively compress models.
Moreover, realizing the actual speedups from compression can be difficult. For example, using unstructured sparsity can lead to nearly 5x improvement in inference efficiency on a CPU while only 1.4x on a GPU.

6.7 FUTURE WORK

Creating robust and efficient language models involves navigating through a large space of configurations from the model size/shape and the training and compression methods. In this thesis, we have enumerated those combinations and explored their impact to understand the trade-offs in efficiency, accuracy, and robustness. In our future work, we seek to improve and expand the work covered in this thesis, apply compressed models to web-scale data, and explore expanded efficiency and robustness.
Improving Compressing Efficiency With regards to existing, our improvements focus on making our approaches in compression more efficient, easier to employ, and more robust. As discussed in 6.6, our existing compression approaches require high degrees of experimentation and many iterations to obtain improved efficiency without major loss in accuracy. We

believe that using improved distillation methods combined with pseudo-labeling methods [218] will likely be a fruitful approach for efficient compression.

Additionally, we seek to combine our work on robust and efficient retrieval to evaluate the trade-offs and interplay in compression and generalization. While we believe representation alignment will be a robust mechanism to merge the two goals, further study is required.

Compression And Web-Scale Data While modeling and evaluation have seen widespread success because of benchmarking datasets [168] [156], the cost of running these models has commonly limited the research on examining model behavior at scale. We believe that by leveraging our approaches in compression on existing and novel benchmarks, we can explore how methods generalize to larger scales. First, we seek to understand how summarization can be used to improve retrieval accuracy for the TREC Deep Learning Track by creating summaries for its 12 million documents. Next, we seek to understand how current state-of-the-art approaches scale to larger corpora. We will explore how models like SPLADE [219] and CoLBERT [174] perform on corpora like ClueWeb 2022 [220].

Multi step optimization via Reinforcement Learning Expanding beyond our existing work, we seek to explore reinforcement learning approaches, which have shown promising results when applied to query execution optimization in databases. Yu et al. '20 [221] have shown that the reinforcement learning approaches can prove effective at query execution optimization, which we believe could extend to optimizing models via layer and operator fusion, pruning, and quantization.

In general, there is an opportunity to apply techniques developed for query optimization to solve the problem of optimizing the compression of language models. While the use of reinforcement learning on language models is not new [222], we believe the scale and uniformity of models would allow for automatic post-training optimization.

As another possibility, we may also adapt analytical cost models [223] to model the overall cost of a neural LM by decomposing it into component costs. While the research focuses on individual systems, large-scale deployments usually comprise many independent systems that execute phase. For example, in modern information retrieval systems having dozens of models run in phases is not uncommon. The compression on end-to-end retrieval has not yet been studied in depth. It is unclear if it is better to compress retrievers such as bi-encoder or re-rankers such as cross-encoders. As a result, further study into how the cost of complex systems can be understood is key to scaling systems.

6.8 PUBLICATIONS

- The Optimal BERT Surgeon: Scalable and Accurate Second-Order Pruning for Large Language Models - Eldar Kurtic, **Daniel Campos**, Tuan Nguyen, Elias Frantar, Mark Kurtz, Benjamin Fineran, Michael Goin, Dan Alistarh - EMNLP 2022

- Sparse*BERT: Sparse Models Generalize To New tasks and Domains - **Daniel Campos**, Alexandre Marques, Tuan Nguyen, Mark Kurtz, ChengXiang Zhai - Sparsity in Neural Networks Workshop at ICML 2022

- oBERTa: Improving Sparse Transfer Learning via improved initialization, distillation, and pruning regimes - **Daniel Campos**, Alexandre Marques, Mark Kurtz, ChengXiang Zhai - The 4th Workshop on Simple and Efficient Natural Language Processing (SustaiNLP 2023) at ACL 2023

- Dense Sparse Retrieval: Using Sparse Language Models for Inference Efficient Dense Retrieval - **Daniel Campos**, ChengXiang Zhai - https://arxiv.org/abs/2304.00114 - Arxiv Preprint

- Noise-Robust Dense Retrieval via Contrastive Alignment Post Training - **Daniel Campos**, Alessandro Magnani, ChengXiang Zhai - https://arxiv.org/abs/2304.03401 - Arxiv Preprint

- Quick Dense Retrievers Consume KALE: Post Training Kullback Leibler Alignment of Embeddings for Asymmetrical dual encoders - **Daniel Campos**, Alessandro Magnani, ChengXiang Zhai - The 4th Workshop on Simple and Efficient Natural Language Processing (SustaiNLP 2023) at ACL 2023

- Compressing Cross-Lingual Multi-task Models at Qualtrics - **Daniel Campos**, Daniel Perry, Samir Joshi, Yashmeet Gambhir, Wei Du, Zhengzheng Xing, and Aaron Colak - The Thirty-Fifth Annual Conference on Innovative Applications of Artificial Intelligence (IAAI-23)

- To Asymmetry and Beyond: Structured Pruning of Sequence to Sequence Models for Improved Inference Efficiency - **Daniel Campos**, ChengXiang Zhai - The 4th Workshop on Simple and Efficient Natural Language Processing (SustaiNLP 2023) at ACL 2023

APPENDIX A: INTRODUCING AND TRANSFERRING SPARSITY FOR EFFICIENT INFERENCE

A.1 THE OPTIMAL BERT SURGEON: SCALABLE AND ACCURATE SECOND-ORDER PRUNING FOR LARGE LANGUAGE MODELS

A.1.1 Additional Comparisons

Here we reflect upon some other methods focused on efficient inference for LLMs, which are orthogonal to weight pruning. For example, Learned Token Pruning [224] tries to remove unimportant tokens in input sequences adaptively and provides 2x higher throughput at ¡ 1% accuracy drop; at the same accuracy drop, our compressed model can achieve 8.4x higher throughput. DeeBERT [115] and FastBERT [225] apply an *early-exit* technique for inference speedup. The latter achieves 2-3x faster inference without performance degradation. However, the method only applies to batch size one. Nevertheless, in terms of direct comparison, our compressed models can achieve 4x faster inference on CPUs without accuracy degradation. Overall, we emphasize that these methods complement our compression techniques, so it would be interesting to investigate computational gains by combining them.

Computational costs In practice, for the 12-layer BERT model with $d = 85M$ encoder weights and block size $B = 50$, the $\mathcal{O}(Bd)$ memory requirement translates to approximately 17GB, which can be easily kept on the 24GB RTX 3090 card. While this amount of memory is available on high-performance GPUs, splitting the $N_B \times B \times B$ tensor along the batch-dimension N_B and utilizing additional GPUs or even memory swapping with CPU is also straightforward. Our implementation quickly updates the inverse Hessian approximation and can run asynchronously while the following gradient is fetched. Computing saliency scores and optimal weight updates take only a few seconds.

A.1.2 Optimal BERT Surgeon (oBERT) Hyper-Parameters

Hyper-parameters. The oBERT pruning method has three tunable hyper-parameters: number of gradients (m), block size (B), and dampening (λ). These are supposed to be tuned for the model and available computational resources. In all of our runs, across all models and datasets, we use the same set of hyper-parameters which we found to work best for the BERT model on the SQuAD v1.1 dataset. We conjecture that further tuning for smaller models (3 and 6-layer models) could improve their results, but for simplicity and fairness to other methods, we apply the same ones found for the BERT.

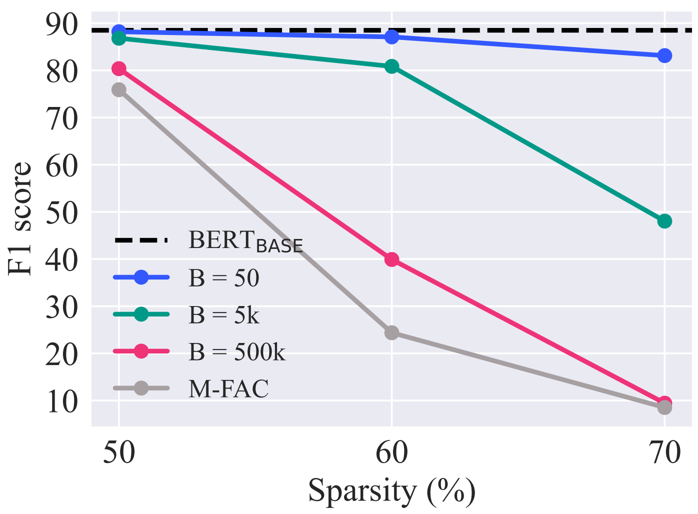

Figure A.1: One-shot pruning ablation study for the block size (B), with $m = 1024$ and $\lambda = 10^{-7}$, on the BERT model and the question-answering SQuAD v1.1 dataset. M-FAC stands for the full inverse Hessian approximation [120].

Ablation studies. The procedure to find the optimal set of hyper-parameters for a model consists of a grid search over the possible hyper-parameter combinations and one-shot pruning runs to various high-sparsity targets to evaluate the quality of the pruning approximation for each combination. We found that $m = 1024$, $B = 50$, and $\lambda = 10^{-7}$ produce state-of-the-art results for a negligible computational overhead with the BERT model. [120] shows that larger block sizes require more gradients for better approximation. Given the size of the BERT model, we picked this setup as it was the best-performing one that could still fit on a single 24GB RTX 3090 GPU card. In Figures A.1, A.2, and A.3, we visualize a fraction of the one-shot pruning ablations concerning all three hyper-parameters that motivated us to pick these specific values.

A.1.3 Downstream Pruning

Teacher preparation. We use the KD from the fine-tuned BERT teacher outputs for all downstream pruning runs. The teacher is fine-tuned on the corresponding downstream task following the default hyper-parameters for SQuAD[44] and GLUE (QQP and MNLI)[45].

Pruning setup. In Table A.1 we describe in detail all hyper-parameters for downstream pruning results presented in Table 3.3. For easier comprehension, we also visualize learning

[44]https://github.com/huggingface/transformers/tree/main/examples/pytorch/question-answering

[45]https://github.com/huggingface/transformers/tree/main/examples/pytorch/text-classification

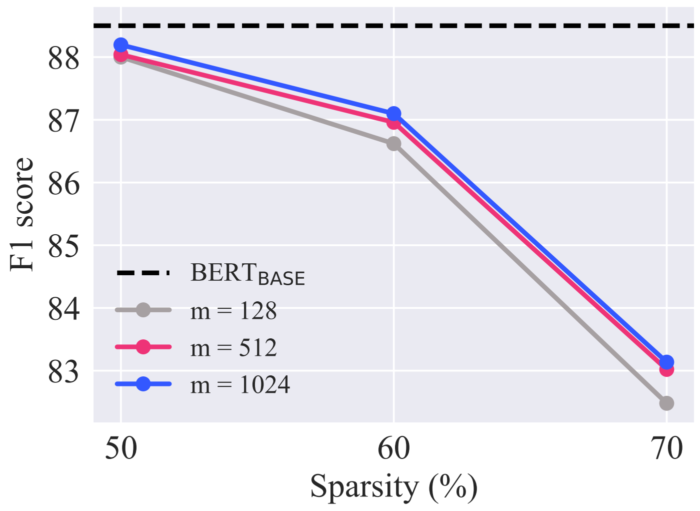

Figure A.2: One-shot pruning ablation study for the number of gradients (m), with $B = 50$ and $\lambda = 10^{-7}$, on the BERT model and the question-answering SQuAD v1.1 dataset.

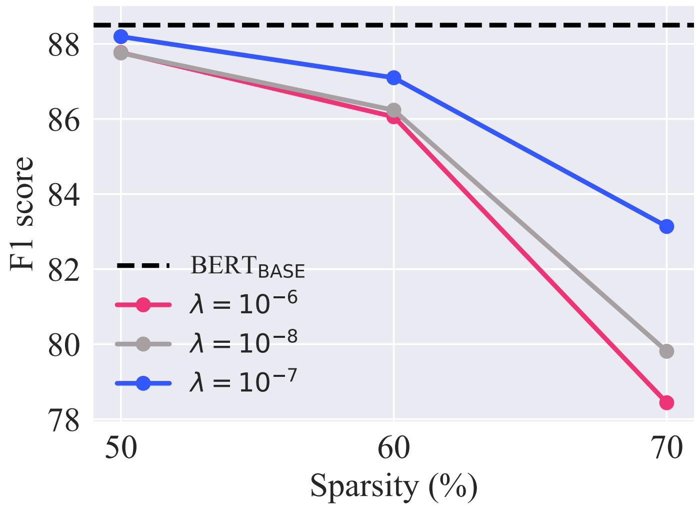

Figure A.3: One-shot pruning ablation study with respect to the dampening (λ), with $m = 1024$ and $B = 50$, on the BERT model and the question-answering SQuAD v1.1 dataset.

rate schedules in Figures A.4 and A.6, and sparsity schedules in Figures A.5 and A.7.

3-, 6-layer models. We prepare our 3 and 6-layer models for downstream runs in layer dropping and retraining phase. We drop layers from our upstream teacher model (more details on it in Appendix A.1.4). After dropping, we retrain the remaining layers, following insights from [129], in the same setup used to prepare the upstream teacher with the addition of the KD from it.

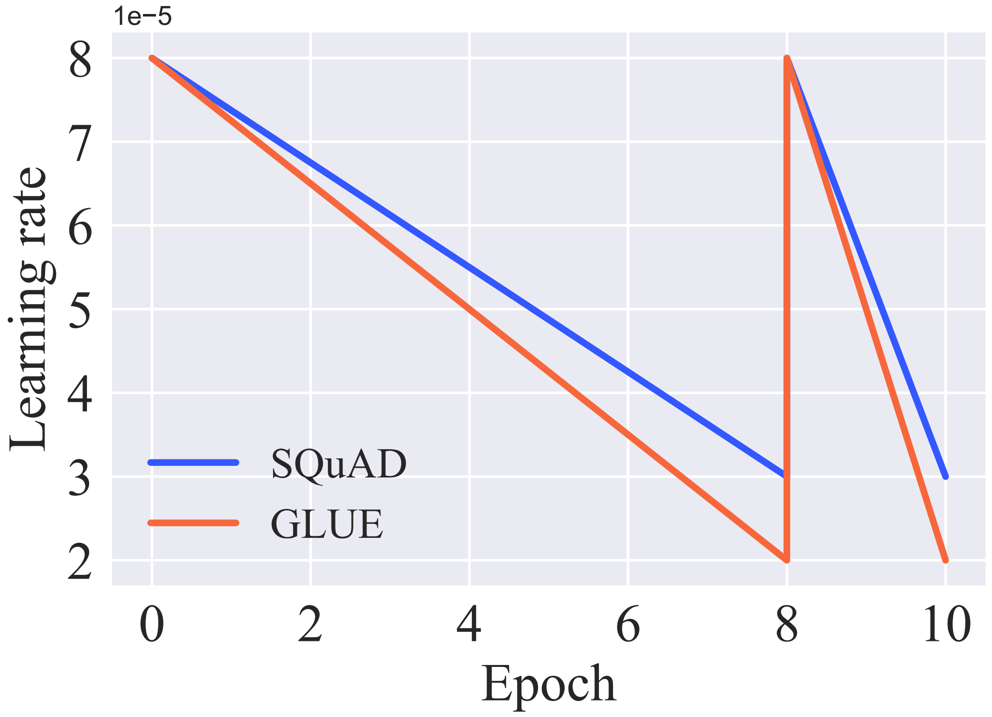

Figure A.4: Visualized learning rate schedule for 10-epoch downstream runs.

A.1.4 Upstream Pruning

Teacher preparation. We prepare a teacher for upstream pruning by following some insights from [51]. More concretely we start with the *bert-base-uncased*[46] model, adopt pre-training on two datasets (BookCorpus[47] & English Wikipedia[48]) with focus on the masked language modeling task (MLM) for 10-epochs with batch size 256 and learning rate linearly decaying to zero from the initial value of 1e-4.

[46]https://huggingface.co/bert-base-uncased
[47]https://huggingface.co/datasets/bookcorpus
[48]https://huggingface.co/datasets/wikipedia

	10 Epochs	30 Epochs
Batch size	16 for SQuAD, 32 for GLUE	
Learning rate (initial, final)	(8e-5, 3e-5) for SQuAD, (8e-5, 2e-5) for GLUE	(8e-5, 8e-6) for SQuAD, (5e-5, 5e-6) for GLUE
Learning rate schedule	linear decay with rewinds	
Learning rate rewinds	one at epoch=8	periodic every 4 epochs, start at epoch=2
Knowledge Distillation (hardness, temp.)	(1.0, 2.0)	
Student model	12-layer: bert-base-uncased 6-layer: layer drop + pre-train with KD 3-layer: layer drop + pre-train with KD	
Teacher model	BERT	
Prune start	epoch=2	
Prune end	epoch=8	epoch=26
Pruning frequency	2x per epoch	once every 4 epochs
Initial sparsity step	12-layer: 70% 6-layer: 30% 3-layer: 30%	
Sparsity distribution	global over all layers	
oBERT parameters	Number of gradients $m = 1024$ Block size $B = 50$ Dampening $\lambda = 10^{-7}$	

Table A.1: Downstream pruning hyper-parameters used to obtain results presented in Tables and 3.3.

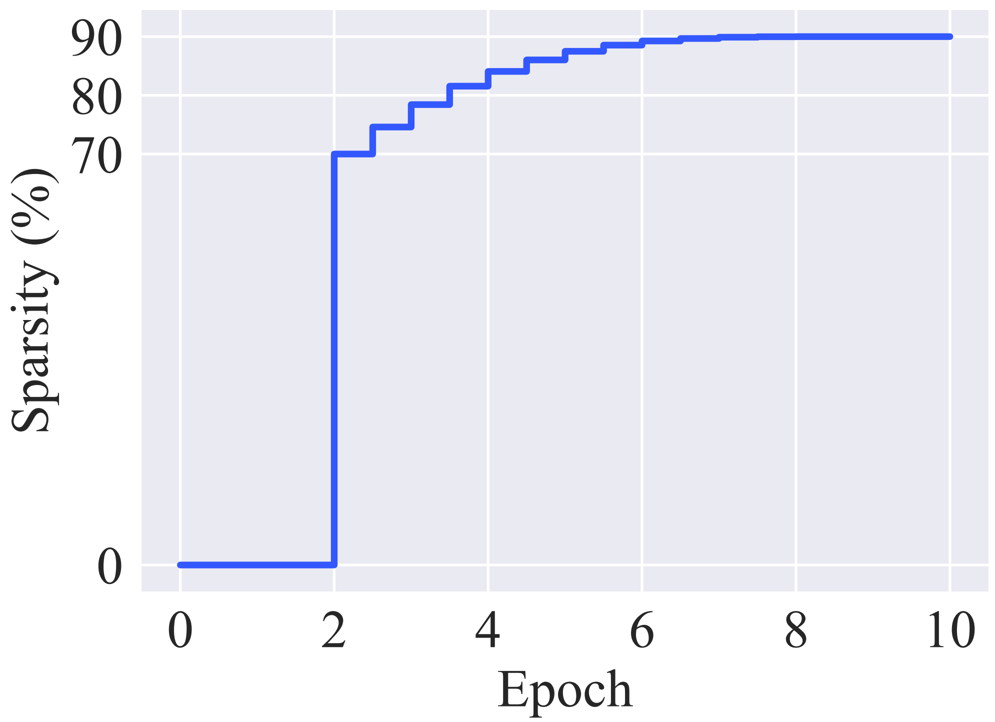

Figure A.5: Visualized sparsity schedule for 10-epoch downstream runs with initial sparsity of 70% and target sparsity of 90%, following the cubic interpolation [128].

Pruning setup. In Table A.15 we describe in detail our upstream pruning recipe. As can be noticed, our upstream pruning recipe is just a downscaled version of our 30-epoch downstream-pruning recipe to 3-epochs.

A.1.5 Downstream Quantization

We perform QAT on top of dense and 4-block pruned models on SQuAD v1.1 as shown in Table 3.3. We quantize to 8 bits the embedding matrices, linear modules of all encoder units which includes matrices in their attention and feed forward layers, and the linear module of the output layer. Weights that were pruned are kept constant (zero) during quantization (sparsity mask preserved). Non-linear operations within the Softmax, LayerNorm and GeLU are not quantized. For each dense and 4-block pruned model in Table 3.3, we perform ten epochs training where the quantization observers are active for the first five and the remaining is fine-tuning. We do hyper-parameter search over the learning rates of 1e-4, 8e-5, 5e-5, 3e-5 and the distillation hardness of 0.9 and 1.0. We then pick the model with the best F1 score.

	3 Epochs
Datasets	BookCorpus & English Wikipedia
Batch size	256
Initial learning rate	5e-4
Learning rate schedule	linear decay with rewinds
Learning rate rewinds	periodic every 0.5 epochs
Max sequence length	512
Weight decay	0.01
Knowledge Distillation (hardness, temperature)	(1.0, 5.5)
Student model	prepared upstream teacher
Teacher model	prepared upstream teacher
Pruning frequency	4x per epoch

Table A.2: Upstream pruning hyper-parameters.

	8 Epochs
Initial learning rate	1.5e-4
Learning rate schedule	linear decay to 1.5e-6
Batch size	16 for SQuAD, 32 for GLUE
Knowledge Distillation (hardness, temperature)	(1.0, 5.5)
Teacher model	BERT

Table A.3: Sparse-transfer learning hyper-parameters used to fine-tune upstream-pruned models at downstream tasks. These hyper-parameters are used to obtain results presented in Table 3.2.

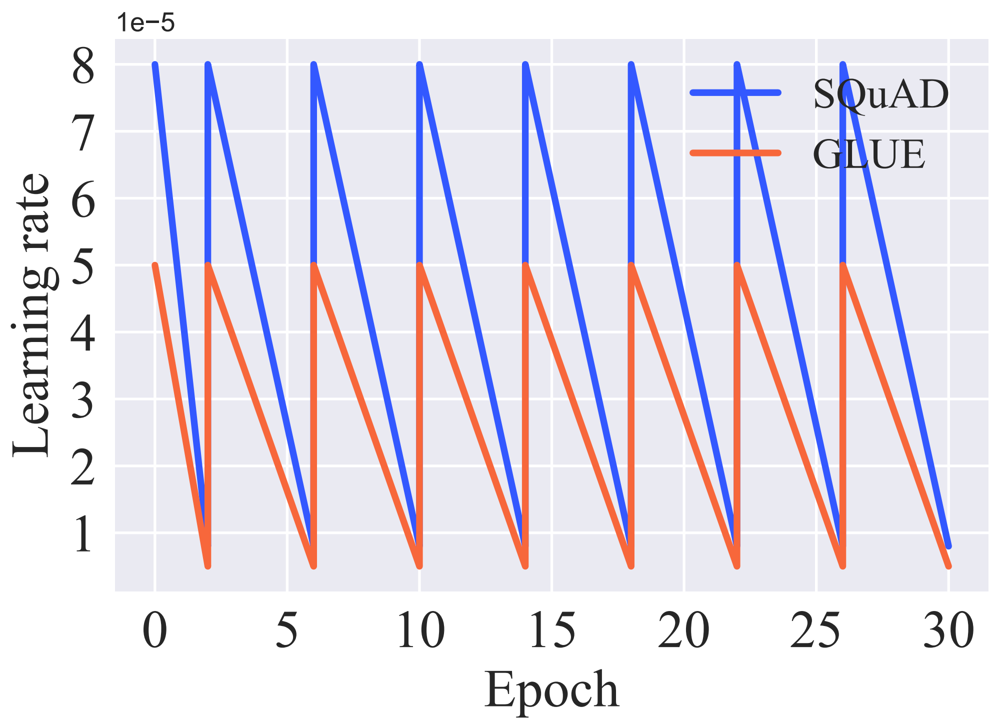

Figure A.6: Visualized learning rate schedule for 30-epoch downstream runs.

A.1.6 Additional performance metrics

Due to the space constraints, in the section we report F1 score for SQuAD v1.1, matched accuracy for MNLI, and accuracy for QQP dataset. As our hyper-parameters for MNLI and QQP are the same, we refer to these two datasets as GLUE. In Table A.4 we report the additional metrics too: exact match (EM) for SQuAD v1.1, mismatched accuracy for MNLI, and F1 score for QQP dataset. Tables A.7 and A.8 present standard deviations of the corresponding results in Tables 3.2 and A.4. Finally, Table A.6 presents the exact-match metric for the corresponding results in Table 3.3.

A.1.7 Inference speedups and compression ratios of compressed models

Details on the results shown in Figure 3.2 are drawn from Table A.9. As shown in the results, not all compound compressed models yield improvements in inference or compression relative to retained model performance but those that do allow for massive improvements.

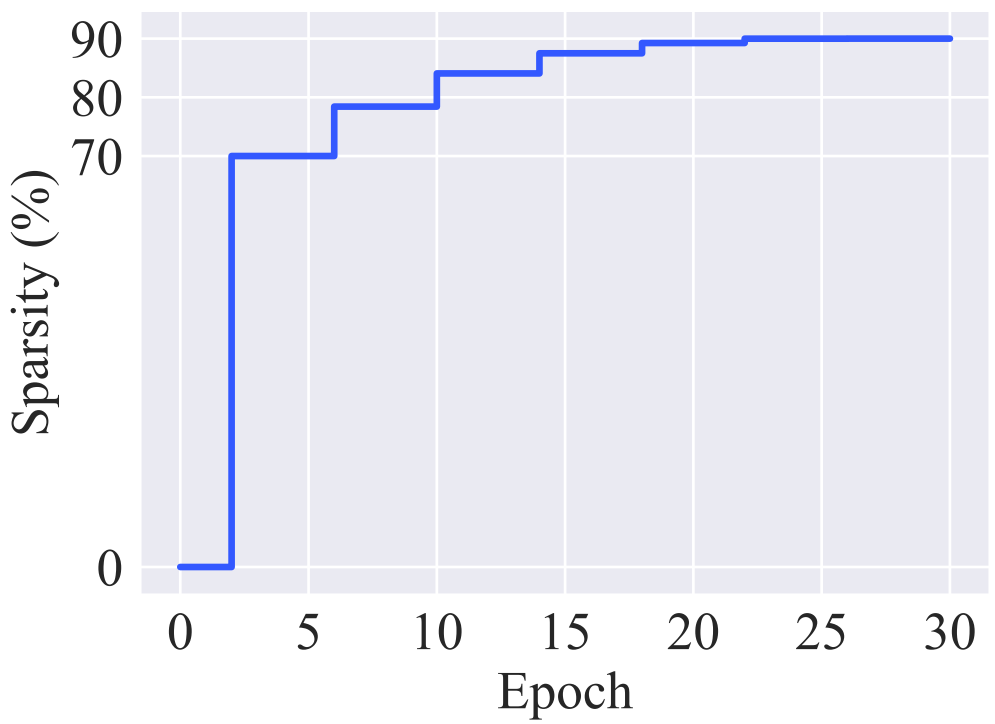

Figure A.7: Visualized sparsity schedule for 30-epoch downstream runs with initial sparsity of 70% and target sparsity of 90%, following the cubic interpolation [128].

Task	BERT BASE	Sparsity	Soft MvP	oBERT (ours)	oBERT (ours)
	Epochs		10 Epochs		30 Epochs
SQuAD EM	81.22	80%	-	-	**82.08**
		90%	76.60	**80.76**	**81.12**
		97%	72.70	**76.14**	**78.11**
MNLI mm-acc	85.06	80%	-	-	**84.91**
		90%	81.80	**83.58**	**84.35**
		97%	80.10	**80.67**	**82.01**
QQP F1	88.00	80%	-	-	**88.63**
		90%	86.80	**87.69**	**88.30**
		97%	85.50	**87.05**	**87.66**

Table A.4: Additional evaluation metrics for results

A.1.8 Responsible NLP Research - Reproducibility Checklist

In addition to many items from the "Reproducibility Checklist" which are already carefully addressed throughout the section and Appendix sections, we provide the remaining

Task	BERT BASE	Sparsity	Prune OFA	oBERT (ours)
SQuAD EM	81.42	90%	79.83	**81.43**
		97%	-	76.90
MNLI mm-acc	85.06	90%	82.43	**83.78**
		97%	-	81.13
QQP F1	88.00	90%	87.72	**87.81**
		97%	-	86.97

Table A.5: Additional evaluation metrics for results presented in Table 3.2.

Layers	Sparsity	Unstructured	4-block	+QAT
12	0%	82.71	82.71	81.99
	80%	82.08	81.46	80.57
	90%	81.12	80.14	78.84
6	0%	81.17	81.17	80.85
	80%	81.15	79.55	78.27
	90%	79.16	77.65	76.56
3	0%	76.62	76.62	76.06
	80%	75.62	74.07	72.70
	90%	73.61	71.36	70.00

Table A.6: Additional evaluation metric (exact-match) for results presented in Table 3.3.

Task	Sparsity	oBERT (ours)
	Epochs	30 Epochs
SQuAD F1, EM	80%	0.11, 0.03
	90%	0.13, 0.13
	97%	0.11, 0.17
MNLI m, mm	80%	0.14, 0.13
	90%	0.05, 0.04
	97%	0.35, 0.22
QQP acc, F1	80%	0.08, 0.08
	90%	0.04, 0.06
	97%	0.05, 0.08

Table A.7: Standard deviations for results presented in Table A.4.

details to facilitate the reproducibility of our results.

Scientific Artifacts Datasets. Our experiments use existing and well-established

Task	Sparsity	oBERT (ours)
SQuAD	90%	0.13, 0.13
F1, EM	97%	0.03, 0.14
MNLI	90%	0.08, 0.24
m, mm	97%	0.17, 0.35
QQP	90%	0.06, 0.07
acc, F1	97%	0.09, 0.18

Table A.8: Standard deviations for results presented in Table 3.2 and A.5.

Layers	Sparsity (%)	Compression Method	F1 score	F1 recall (%)	Throughput (items per sec.)	Speedup DeepSparse	Model size (gzip MB)	Compression Ratio (w.r.t. gzip)
12	0	none	88.54	100.00	65.81	1.00	384.7	1.00
12	80	unstructured	89.04	100.56	222.66	3.38	173.1	2.22
12	90	unstructured	88.31	99.74	292.40	4.44	140.1	2.75
12	80	4-block+QAT	87.89	99.26	552.22	8.39	37.8	10.18
6	80	unstructured	88.20	99.62	419.68	6.38	128.3	3.00
6	90	unstructured	86.78	98.01	663.02	10.07	111.8	3.44
6	80	4-block+QAT	86.10	97.24	989.54	15.04	26.2	14.70
3	80	unstructured	84.08	94.96	737.62	11.21	105.9	3.63
3	90	unstructured	82.50	93.18	974.00	14.80	97.7	3.94
3	80	4-block+QAT	82.04	92.66	1892.27	28.75	20.3	18.92

Table A.9: Compression effects on model size and inference speed, evaluated at batch size 32 with sequence length 128 on SQuAD v1.1 dataset. Evaluated at the c5.12xlarge AWS instance.

benchmarks for pre-training and fine-tuning LLMs. Each dataset was used without any additional forms of modifications. Given that we did not modify any datasets, we did not inspect for personal, sensitive, or offensive content or perform any anonymization. For pre-training, we make use of the Toronto Book Corpus (TBC) [226] [49] and the wikipedia.20200501.en [143] [50]. For fine-tuning we make use of SQuAD v1.1 [7] [51], Quora Duplicate Question Dataset (QQP) [52], and Multi-Genre Natural Language Inference (MNLI) [127] [53] datasets. All these datasets are publicly available via HuggingFace datasets repository [125]. The terms of usage and further details on each dataset can be found in their respective repositories.

Models. The model used as a starting point for all of our experiments is BERT, publicly available via HuggingFace Hub [54]. All other models presented in this section will be released

[49]https://huggingface.co/datasets/bookcorpus
[50]https://huggingface.co/datasets/wikipedia
[51]https://huggingface.co/datasets/squad
[52]https://huggingface.co/datasets/glue
[53]https://huggingface.co/datasets/glue
[54]https://huggingface.co/bert-base-uncased

in openly-available repositories along with their compression recipes, training metrics, and hyper-parameters.

Dataset Statistics Dataset statistics are detailed in Table A.12.

Dataset	Train	Eval
SQuAD (examples)	87599	10570
MNLI (examples)	392702	19628
QQP (examples)	363,846	40,430
Wikipedia (words)	6078422	-
TBC (words)	74004228	-

Table A.10: Statistics for training and evaluation datasets

A.1.9 Computational Experiments

Upstream. All upstream runs are, in general, computationally expensive due to the large batch sizes and huge datasets. In our experiments, we use 4x A100 40GB NVIDIA GPUs. In this configuration, a single training epoch takes approximately 6 hours. Since the cost of such a large compute instance is high, these experiments were only run with a single seed and without major hyper-parameter exploration.

Downstream. Our downstream experiments use various GPU cards: 16GB V100, 11GB RTX 2080 Ti, and 24GB RTX 3090. Each training epoch takes approximately 30 minutes, so the 30 epoch runs take approximately 15 hours. We report the mean results of three runs with different random seeds for these experiments.

DeepSparse inference. We pair our compressed models with DeepSparse [119], a publicly-available sparsity-aware CPU inference engine. This CPU runtime can leverage structured and unstructured sparsity and quantization to deliver high performance on commodity CPUs. We ran DeepSparse on a 24-core Intel AWS c5.12xlarge server with 24 cores, 96 vCPUs, 192 GB of RAM, and an AVX-512 compatible instruction set. All models are exported using the standard ONNX[55] format.

A.1.10 Computational Packages

Our experiments build on publicly available libraries to ensure ease of reproduction and extensibility. All of our implementations, training, and evaluation code are built on top of

[55]https://onnx.ai/

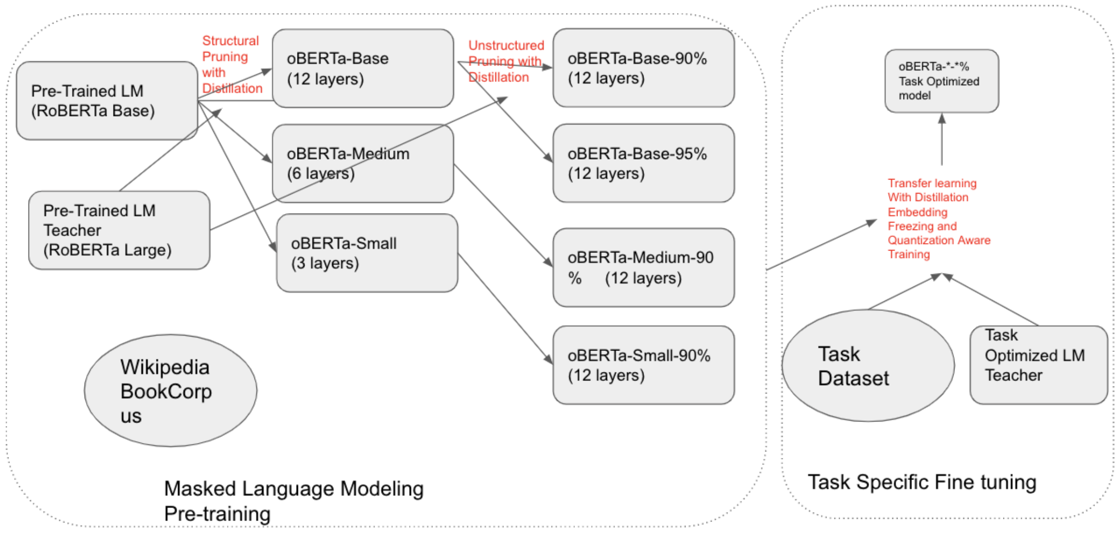
Pre-Trained LM (RoBERTa Base)
Structural Pruning with Distillation
oBERTa-Base (12 layers)
Unstructured Pruning with Distillation
oBERTa-Base-90% (12 layers)
oBERTa-Base-95% (12 layers)
oBERTa-Medium (6 layers)
oBERTa-Medium-90 % (12 layers)
oBERTa-Small (3 layers)
oBERTa-Small-90% (12 layers)
Pre-Trained LM Teacher (RoBERTa Large)
Wikipedia BookCorp us
Masked Language Modeling Pre-training
oBERTa-*-*% Task Optimized model
Transfer learning With Distillation Embedding Freezing and Quantization Aware Training
Task Dataset
Task Optimized LM Teacher
Task Specific Fine tuning

Figure A.8: The set of oBERTa language models follows a compounding compression approach. First models are structurally pruned and further pre-trained using KD and a RoBERTa$_{\text{large}}$ teacher. Next, each model is pruned during additional pre-training to a target sparsity. After pruning, the sparsity pattern is locked, and models are fine-tuned with KD on specialized NLP tasks. During fine-tuning, models may be quantized for additional improvements in inference efficiency.

HuggingFace's Transformers [56] and Datasets [57] libraries, NeuralMagic's SparseML [58] library for model compression, and their DeepSparse [59] engine for efficient inference on commodity CPUs.

A.2 OBERTA: IMPROVING SPARSE TRANSFER LEARNING VIA IMPROVED INITIALIZATION, DISTILLATION, AND PRUNING REGIMES

A.2.1 Model Generation Approach

oBERTa models are generated in a multi-stage approach with details found in figure A.8

[56]https://github.com/huggingface/transformers
[57]https://github.com/huggingface/datasets
[58]https://github.com/neuralmagic/sparseml
[59]https://github.com/neuralmagic/deepsparse

Model	Parameters	Prunable	Sparse	Sparsity	size (MB)	Compression	GZIP size (MB)	Compression
oBERTa$_{\text{base}}$	124,647,170	85,526,016	1,539	0.0%	474	1.00	435	1.00
oBERTa$_{\text{base}}$ Quantized	124,647,170	85,526,016	1,539	0.0%	119	3.98	85	5.12
oBERTa$_{\text{base}}$ 90%	124,647,170	85,526,016	76,442,738	89.4%	474	1.00	183	2.38
oBERTa$_{\text{base}}$ 90% Quantized	124,647,170	85,526,016	76,442,738	89.4%	119	3.98	42	10.36
oBERTa$_{\text{base}}$ 95%	124,647,170	85,526,016	80,689,466	94.3%	474	1.00	163	2.67
oBERTa$_{\text{base}}$ 95% Quantized	124,647,170	85,526,016	80,689,466	94.3%	119	3.98	37	11.76
oBERTa$_{\text{MEDIUM}}$	82,119,938	43,058,688	1,538	0.0%	312	1.52	289	1.51
oBERTa$_{\text{MEDIUM}}$ Quantized	82,119,938	43,058,688	1,538	0.0%	78	6.08	53	8.21
oBERTa$_{\text{MEDIUM}}$ 90%	82,119,938	43,058,688	38,222,138	88.8%	312	1.52	161	2.70
oBERTa$_{\text{MEDIUM}}$ 90% Quantized	82,119,938	43,058,688	38,222,138	88.8%	78	6.08	33	13.18
oBERTa$_{\text{SMALL}}$	60,856,322	21,825,024	1,538	0.0%	233	2.03	214	2.03
oBERTa$_{\text{SMALL}}$ Quantized	60,856,322	21,825,024	1,538	0.0%	60	7.90	39	11.15
oBERTa$_{\text{SMALL}}$ 90%	60,856,322	21,825,024	19,111,068	87.6%	233	2.03	149	2.92
oBERTa$_{\text{SMALL}}$ 90% Quantized	60,856,322	21,825,024	19,111,838	87.6%	60	7.90	30	14.50

Table A.11: Description of the oBERTa model family and their sparsity and size. Prunable parameters are the sum of all non-embedding parameters in the model. Since sparsity profiles are assigned at a module level, overall sparsity profiles do not perfectly match the target 90% or 95% which are targeted.

A.2.2 Roberta and Training Methodology

RoBERTa [51] is a language model that can best be considered more robust and optimized for the popular BERT model. While the models share architectures, their training differs as RoBERTA uses a 160 GB corpus for 10 epochs compared to the 4GB one used by BERT. As a result, the training time of RoBERTA is about 100 times higher than its predecessor.
Given this high cost of training and the regular need for longer training when pruning a model [138], we focus on compressing RoBERTa without following its expensive pre-training regime. Our research leverages the popular open-source compression library SparseML[60] to implement unstructured pruning, structured pruning, and quantization via quantization-aware training. In all our experiments, we prune each network component independently using GMP or Optimal BERT Surgeon (OBS) (Kurtic et al.). One exception is the embeddings layer, which we do not prune.

A.2.3 Model Details

Model details can be found in table A.11

A.2.4 Dataset Details

Dataset statistics are detailed in Table A.12.

[60]https://github.com/neuralmagic/sparseml

Dataset	Train	Eval
SQuAD v1.1 (examples)	87599	10570
SQuAD v2.0 (examples)	130319	11873
MNLI (examples)	392702	19628
QQP (examples)	363,846	40,430
IMDB (examples)	25000	25000
CONLL2003 (examples)	14041	3250
SST2 (examples)	67349	872
Wikipedia (words)	6078422	-
TBC (words)	74004228	-

Table A.12: Statistics for training and evaluation datasets

A.2.5 Teacher Models

Performance of the RoBERTa$_{\text{base}}$and RoBERTa$_{\text{large}}$ models on our sparse transfer datasets. We explore the optimal hyperparameters relative to performance in published results as shown in table A.13 and A.14

Model	Training Epochs	Batch Size	Learning Rate	Weight Decay	Warmup	Target Metric	Target Score	Actual	Recall
SQUAD V1.1	3	16	1.00E-05	0	0	F1	90.40	92.15	101.94%
SQUAD V2.0	3	16	3.00E-05	0	0	F1	82.91	83.53	100.74%
QQP	5	16	2.00E-05	0	0	ACC	91.90	91.52	99.59%
MNLI	3	16	1.00E-05	0	0	ACC	87.60	87.88	100.31%
SST-2	3	16	2.00E-05	0	0	ACC	94.80	94.61	99.80%
CONLL2003	3	16	3.00E-05	0	0	ACC	99.10	99.29	100.19%
IMDB	3	16	1.00E-05	0	0	ACC	94.67	95.24	100.60%

Table A.13: Training parameters along with performance metrics and the recovery vs. the published performance of the same model for the RoBERTa base model

Model	Training Epochs	Batch Size	Learning Rate	Weight Decay	Warmup	Target Metric	Target Score	Actual	Recall
SQUAD V1.1	3	16	1.00E-05	0	0	F1	94.50	94.62	100.12%
SQUAD V2.0	3	16	1.00E-05	0	0	F1	89.40	89.14	99.71%
QQP	3	16	1.00E-05	0	0	ACC	92.20	91.76	99.52%
MNLI	3	16	1.00E-05	0	0	ACC	90.20	90.61	100.45%
SST-2	3	16	1.00E-05	0	0	ACC	96.40	96.22	99.81%
CONLL2003	3	16	3.00E-05	0	0	ACC	99.10	99.39	100.29%
IMDB	3	16	1.00E-05	0	0	ACC	94.67	96.12	101.53%

Table A.14: Training parameters along with performance metrics and the recovery vs. the published performance of the same model for the RoBERTa large model

	5 Epochs
Datasets	BookCorpus & English Wikipedia
Batch size	256
Initial learning rate Learning rate schedule Learning rate rewinds	5e-4 linear decay with rewinds periodic every 0.5 epochs
Max sequence length Weight decay	512 0.01
Knowledge Distillation (hardness, temperature)	(1.0, 5.5)
Student model Teacher model	dense oBERTa-* model RoBERTa$_{\text{large}}$
Pruning frequency	100x per epoch
Initial Sparsity	0.7 for 12 layer model, 0.5 for the 6-layer, and 0.3 for the 3-layer

Table A.15: Upstream pruning hyper-parameters.

A.2.6 Upstream Pruning

Following the findings that more extensive teachers distill better [51] and our experiments, we use both RoBERTa$_{\text{base}}$and RoBERTa$_{\text{large}}$ as teachers eventually find the large model works better. Using this teacher, we use the parameters shown in table A.15 to prune the models for oBERTa. This same set of parameters is applied to the structurally pruned models, but there is no induced sparsity.

A.2.7 Sparse Transfer Hyper-Parameters

Our work aims not to produce the highest possible performance of a sparse language model. Instead, we aim to make light language models that perform well on various tasks with minimal hyperparameter optimization. As a result, in all of our experiments, we leverage the parameters shown in C.2 and A.17 and perform a grid search over them.

A.2.8 Learning Rate

In our exploration of sparse transfer learning, we perform a wide study on the impact of the optimal learning rate for each task and each model in the oBERTa family. The results as shown in table A.18

	10 Epochs
Initial learning rate Learning rate schedule	2.1e-4,1.9e-4,1.7e-4,1.5e-4,1.3e-4,1.1e-4,9e-5,7e-5,5e-5,3e-5,2e-5,1e-5 linear decay to 0
Batch size	12
Weight Decay	0.0, 0.01, 0.05, 0.1
Knowledge Distillation hardness	1.0, 0.0
Frozen Embeddings	1.0, 0.0
Knowledge Distillation temperature	7.0
Knowledge Distillation Teacher	RoBERTa$_{\text{base}}$, RoBERTa$_{\text{large}}$

Table A.16: Sparse-transfer learning hyper-parameters used to fine-tune upstream-pruned models at downstream tasks. Each Experiment tunes this set of parameters to find a task-specific optimal combination.

	20 Epochs
Initial learning rate Learning rate schedule	2.1e-4,1.9e-4,1.7e-4,1.5e-4,1.3e-4,1.1e-4,9e-5,7e-5,5e-5,3e-5,2e-5,1e-5 linear decay to 0. Rewind to 5e-5 for QAT at epoch 10
Freeze Batch Norm Epoch	18
Batch size	12
Weight Decay	0.0, 0.01, 0.05, 0.1
Knowledge Distillation hardness	1.0, 0.0
Frozen Embeddings	1.0, 0.0
Frozen Embeddings Schedule	Frozen until epoch 10, unfrozen for QAT
Knowledge Distillation temperature	7.0
Knowledge Distillation Teacher	RoBERTa$_{\text{base}}$, RoBERTa$_{\text{large}}$

Table A.17: Sparse-transfer learning with Quantization hyper-parameters used to fine-tune upstream-pruned models at downstream tasks. Each Experiment tunes this set of parameters to find a task-specific optimal combination.

	Optimal Learning Rate						
model	SQUAD	SQUAD V2	MNLI	QQP	IMDB	SST2	CONLL2003
RoBERTa$_{\text{base}}$	1.00E-05	3.00E-05	1.00E-05	2.00E-05	1.00E-05	2.00E-05	3.00E-05
RoBERTa$_{\text{large}}$	1.00E-05	1.00E-05	1.00E-05	1.00E-05	1.00E-05	1.00E-05	3.00E-05
oBERTa$_{\text{base}}$	1.00E-05	1.00E-05	1.00E-05	2.00E-05	1.00E-05	2.00E-05	3.00E-05
oBERTa$_{\text{base}}$ 90%	1.50E-04	1.50E-04	7.00E-05	1.70E-04	1.30E-04	9.00E-05	1.50E-04
oBERTa$_{\text{base}}$ 95%	1.50E-04	1.30E-04	9.00E-05	2.10E-04	1.30E-04	9.00E-05	5.00E-05
oBERTa$_{\text{MEDIUM}}$	5.00E-05	5.00E-05	2.00E-05	3.00E-05	3.00E-05	2.00E-05	3.00E-05
oBERTa$_{\text{MEDIUM}}$ 90%	1.50E-04	1.30E-04	1.50E-04	1.50E-04	5.00E-05	1.50E-04	1.50E-04
oBERTa$_{\text{SMALL}}$	1.50E-04	1.50E-04	3.00E-05	5.00E-05	3.00E-05	5.00E-05	3.00E-05
oBERTa$_{\text{SMALL}}$ 90%	1.50E-04	1.50E-04	2.10E-04	2.10E-04	1.50E-04	2.10E-04	1.90E-04

Table A.18: Sparse-transfer learning with Quantization hyper-parameters used to fine-tune upstream-pruned models at downstream tasks. Each Experiment tunes this set of parameters to find a task-specific optimal combination.

A.2.9 Knowledge Distillation

In our exploration of sparse transfer learning, we perform a wide study on the impact of knowledge distillation. Across tasks, we look at the impact using no teacher, RoBERTa$_{\text{base}}$and RoBERTa$_{\text{large}}$ as shown in tables A.19,A.20,A.21,A.22,A.23,A.24

model	No KD	KD-Base	KD-Large
oBERTa$_{\text{base}}$(Target)	91.52%	N/A	N/A
oBERTa$_{\text{base}}$ 90%	91.97	92.78	92.55
oBERTa$_{\text{base}}$ 95%	91.40	91.17	91.514
oBERTa$_{\text{MEDIUM}}$	90.94	91.86	91.78
oBERTa$_{\text{MEDIUM}}$ 90%	87.16	87.16	89.56
oBERTa$_{\text{SMALL}}$	89.56	88.65	90.83
oBERTa$_{\text{SMALL}}$ 90%	85.58	89.22	89.45

Table A.19: Impact of knowledge distillation on the accuracy (matched) MNLI Dataset across model sizes for the various sizes of oBERTa as compared to the regularly trained baseline

model	No KD	KD-Base	KD-Large
oBERTa$_{\text{base}}$(Target)	91.52	N/A	N/A
oBERTa$_{\text{base}}$ 90%	63.18	91.01	90.93
oBERTa$_{\text{base}}$ 95%	90.46	90.45	90.72
oBERTa$_{\text{MEDIUM}}$	90.75	90.96	90.96
oBERTa$_{\text{MEDIUM}}$ 90%	89.93	90.41	89.82
oBERTa$_{\text{SMALL}}$	86.63	87.34	87.65
oBERTa$_{\text{SMALL}}$ 90%	88.72	89.40	87.50

Table A.20: Impact of knowledge distillation on the accuracy QQP Dataset across model sizes for the various sizes of oBERTa as compared to the regularly trained baseline

model	No KD	KD-Base	KD-Large
oBERTa$_{\text{base}}$(Target)	91.52	N/A	N/A
oBERTa$_{\text{base}}$ 90%	91.97	92.78	92.55
oBERTa$_{\text{base}}$ 95%	91.4	91.17	91.514
oBERTa$_{\text{MEDIUM}}$	90.94	91.86	91.78
oBERTa$_{\text{MEDIUM}}$ 90%	87.16	87.16	89.56
oBERTa$_{\text{SMALL}}$	89.56	88.65	90.83
oBERTa$_{\text{SMALL}}$ 90%	85.58	89.22	89.45

Table A.21: Impact of knowledge distillation on the accuracy SST-2 Dataset across model sizes for the various sizes of oBERTa as compared to the regularly trained baseline

model	No KD	KD-Base	KD-Large
oBERTa$_{\text{base}}$(Target)	91.52%	N/A	N/A
oBERTa$_{\text{base}}$ 90%	99.17	99.08	99.11
oBERTa$_{\text{base}}$ 95%	98.89	98.47	97.51
oBERTa$_{\text{MEDIUM}}$	99.21	99.16	99.19
oBERTa$_{\text{MEDIUM}}$ 90%	99.01	98.8	98.79
oBERTa$_{\text{SMALL}}$	99.05	98.95	98.94
oBERTa$_{\text{SMALL}}$ 90%	98.88	98.55	98.55

Table A.22: Impact of knowledge distillation on the accuracy on the CONLL2003 Dataset across model sizes for the various sizes of oBERTa as compared to the regularly trained baseline

model	No KD	KD-Base	KD-Large
oBERTa$_{\text{base}}$(Target)	91.52%	N/A	N/A
oBERTa$_{\text{base}}$ 90%	89.01	90.86	90.92
oBERTa$_{\text{base}}$ 95%	87.06	89.84	89.21
oBERTa$_{\text{MEDIUM}}$	84.36	88.20	85.74
oBERTa$_{\text{MEDIUM}}$ 90%	84.71	89.26	88.61
oBERTa$_{\text{SMALL}}$	82.00	80.77	77.08
oBERTa$_{\text{SMALL}}$ 90%	73.31	84.66	83.13

Table A.23: Impact of knowledge distillation on the F1 SQUAD v1.1 Dataset across model sizes for the various sizes of oBERTa as compared to the regularly trained baseline

model	No KD	KD-Base	KD-Large
oBERTa$_{\text{base}}$(Target)	91.52%	N/A	N/A
oBERTa$_{\text{base}}$ 90%	75.57852204	80.25256971	81.32561567
oBERTa$_{\text{base}}$ 95%	72.61	77.67	77.98
oBERTa$_{\text{MEDIUM}}$	69.42634	70.97328	71.55996
oBERTa$_{\text{MEDIUM}}$ 90%	68.25281	76.02975	76.64135
oBERTa$_{\text{SMALL}}$	66.8281	62.9573	63.1224
oBERTa$_{\text{SMALL}}$ 90%	55.3959	70.0796	70.7913

Table A.24: Impact of knowledge distillation on the F1 SQUAD v2.0 Dataset across model sizes for the various sizes of oBERTa as compared to the regularly trained baseline

A.2.10 Freezing Embeddings

In our exploration of sparse transfer learning, we perform a wide study on the impact of freezing the embeddings during finetuning. Across tasks, we look at the impact of frozen and unfrozen embeddings as shown in tables A.25,A.26,A.27,A.28,A.29, and A.30. Besides question answering, we do not find a strong trend with the impact of frozen embeddings. In some tasks, sparse and dense models perform better with frozen embeddings while not for others. Focusing on question answering, by using frozen embeddings dense models see large losses in F1 score and the opposite can be seen for pruned models.

model	Frozen	Unfrozen
oBERTa$_{\text{base}}$ (Target)	N/A	87.88%
oBERTa$_{\text{base}}$ 90%	84.50	83.81
oBERTa$_{\text{base}}$ 95%	83.91	83.41
oBERTa$_{\text{MEDIUM}}$	84.37	83.32
oBERTa$_{\text{MEDIUM}}$ 90%	81.61	77.00
oBERTa$_{\text{SMALL}}$	80.24	80.36
oBERTa$_{\text{SMALL}}$ 90%	78.46	74.25

Table A.25: Impact of frozen vs trained embeddings on the accuracy (matched) MNLI Dataset across model sizes for the various sizes of oBERTa as compared to the uncompressed baseline

model	Frozen	Unfrozen
oBERTa$_{\text{base}}$ (Target)	N/A	91.52%
oBERTa$_{\text{base}}$ 90%	90.93%	90.99%
oBERTa$_{\text{base}}$ 95%	90.72%	90.85%
oBERTa$_{\text{MEDIUM}}$	90.96%	91.35%
oBERTa$_{\text{MEDIUM}}$ 90%	89.82%	90.48%
oBERTa$_{\text{SMALL}}$	90.59%	90.72%
oBERTa$_{\text{SMALL}}$ 90%	89.40%	89.74%

Table A.26: Impact of frozen vs trained embeddings on the accuracy on QQP across model sizes for the various sizes of oBERTa as compared to the uncompressed baseline

model	Frozen	Unfrozen
oBERTa$_{\text{base}}$ (Target)	N/A	91.52%
oBERTa$_{\text{base}}$ 90%	92.55	91.74
oBERTa$_{\text{base}}$ 95%	91.514	91.4
oBERTa$_{\text{MEDIUM}}$	91.78	92.89
oBERTa$_{\text{MEDIUM}}$ 90%	89.56	88.76
oBERTa$_{\text{SMALL}}$	90.83	90.48
oBERTa$_{\text{SMALL}}$ 90%	89.45	89.34

Table A.27: Impact of frozen vs trained embeddings on the accuracy SST2 Dataset across model sizes for the various sizes of oBERTa as compared to the uncompressed baseline

A.2.11 Inference Benchmarks

We provide full results for our experiments in benchmarking the impact of compression on inference efficiency as shown in tables A.38,A.36,A.35,A.31,A.33,A.32,A.37,A.37

model	Frozen	Unfrozen
oBERTa$_{\text{base}}$ (Target)	N/A	91.52%
oBERTa$_{\text{base}}$ 90%	97.51	98.55
oBERTa$_{\text{base}}$ 95%	99.11	99.13
oBERTa$_{\text{MEDIUM}}$	99.19	99.18
oBERTa$_{\text{MEDIUM}}$ 90%	98.79	98.9
oBERTa$_{\text{SMALL}}$	98.94	98.94
oBERTa$_{\text{SMALL}}$ 90%	98.55	98.69

Table A.28: Impact of frozen vs trained embeddings on the accuracy on CONLL2003 Dataset across model sizes for the various sizes of oBERTa as compared to the uncompressed baseline

model	Frozen	Unfrozen
oBERTa$_{\text{base}}$ (Target)	N/A	91.52%
oBERTa$_{\text{base}}$ 90%	90.92	83.99
oBERTa$_{\text{base}}$ 95%	89.21	87.08
oBERTa$_{\text{MEDIUM}}$	85.74	89.95
oBERTa$_{\text{MEDIUM}}$ 90%	88.61	86.63
oBERTa$_{\text{SMALL}}$	77.08	84.64
oBERTa$_{\text{SMALL}}$ 90%	83.13	77.43

Table A.29: Impact of frozen vs trained embeddings on SQUAD v1.1 F1 across model sizes for the various sizes of oBERTa as compared to the uncompressed baseline

model	Frozen	Unfrozen
oBERTa$_{\text{base}}$ (Target)	N/A	91.52%
oBERTa$_{\text{base}}$ 90%	71.56	78.05
oBERTa$_{\text{base}}$ 95%	81.33	78.45
oBERTa$_{\text{MEDIUM}}$	77.98	76.86
oBERTa$_{\text{MEDIUM}}$ 90%	76.64	72.77
oBERTa$_{\text{SMALL}}$	71.32	63.12
oBERTa$_{\text{SMALL}}$ 90%	70.79	59.38

Table A.30: Impact of frozen vs trained embeddings on the SQUAD v2.0 Dataset across model sizes for the various sizes of oBERTa as compared to the uncompressed baseline

A.2.12 Limitations

While much of our work has focused on showcasing the broad usability of compressed language models, they are not without fault. While our experiments focus on the compression of RoBERTa, the size of its training dataset makes complete exploration of the ability of pruning during pretraining somewhat limited. The work in the paper shows the ability to compress RoBERTa on a smaller pretraining dataset but does not contrast it with the impact

model	Throughput (items/sec)	Speedup	Latency Mean (ms/batch)	Latency Median (ms/batch	Latency Std (ms/batch)
oBERTa$_{\text{base}}$	16.69	1.00	59.90	59.82	1.02
oBERTa$_{\text{base}}$ Quantized	51.68	3.10	19.34	19.28	0.58
oBERTa$_{\text{base}}$ 90%	54.87	3.29	18.21	18.15	0.31
oBERTa$_{\text{base}}$ 90% Quantized	68.70	4.12	14.55	14.50	0.20
oBERTa$_{\text{base}}$ 95%	145.57	8.72	6.86	6.86	0.11
oBERTa$_{\text{base}}$ 95% Quantized	78.90	4.73	12.66	12.68	0.31
oBERTa$_{\text{MEDIUM}}$	32.78	1.96	30.49	30.44	1.19
oBERTa$_{\text{MEDIUM}}$ Quantized	103.47	6.20	9.65	9.60	0.57
oBERTa$_{\text{MEDIUM}}$ 90%	106.01	6.35	9.42	9.34	0.28
oBERTa$_{\text{MEDIUM}}$ 90% Quantized	149.25	8.94	6.69	6.65	0.42
oBERTa$_{\text{SMALL}}$	64.93	3.89	15.39	15.31	0.66
oBERTa$_{\text{SMALL}}$ Quantized	208.09	12.47	4.80	4.78	0.28
oBERTa$_{\text{SMALL}}$ 90%	203.95	12.22	4.89	4.86	0.33
oBERTa$_{\text{SMALL}}$ 90% Quantized	270.63	16.21	3.69	3.68	0.25

Table A.31: Inference performance of the oBERTa model family using a batch size of 1, 24 cores, and a sequence length of 384

model	Throughput (items/sec)	Speedup	Latency Mean (ms/batch)	Latency Median (ms/batch	Latency Std (ms/batch)
oBERTa$_{\text{base}}$	19.55	1.00	818.23	811.93	15.52
oBERTa$_{\text{base}}$ Quantized	83.92	4.29	190.65	189.55	4.21
oBERTa$_{\text{base}}$ 90%	74.29	3.80	215.35	214.31	2.47
oBERTa$_{\text{base}}$ 90% Quantized	137.83	7.05	116.07	115.43	2.56
oBERTa$_{\text{base}}$ 95%	89.07	4.56	179.62	178.92	3.19
oBERTa$_{\text{base}}$ 95% Quantized	160.68	8.22	99.56	98.91	2.63
oBERTa$_{\text{MEDIUM}}$	38.95	1.99	410.73	408.13	6.11
oBERTa$_{\text{MEDIUM}}$ Quantized	157.12	8.04	101.82	101.27	2.21
oBERTa$_{\text{MEDIUM}}$ 90%	144.95	7.41	110.37	109.62	1.56
oBERTa$_{\text{MEDIUM}}$ 90% Quantized	251.32	12.86	63.65	63.40	1.76
oBERTa$_{\text{SMALL}}$	77.49	3.96	206.46	205.75	2.07
oBERTa$_{\text{SMALL}}$ Quantized	276.10	14.12	57.94	57.43	1.63
oBERTa$_{\text{SMALL}}$ 90%	281.57	14.40	56.81	56.73	0.64
oBERTa$_{\text{SMALL}}$ 90% Quantized	417.35	21.35	38.32	38.01	1.55

Table A.32: Inference performance of the oBERTa model family using a batch size of 16, 24 cores, and a sequence length of 384

model	Throughput (items/sec)	Speedup	Latency Mean (ms/batch)	Latency Median (ms/batch	Latency Std (ms/batch)
oBERTa$_{\text{base}}$	19.02	1.00	3365.11	3352.63	29.49
oBERTa$_{\text{base}}$ Quantized	84.80	4.46	754.73	749.38	18.69
oBERTa$_{\text{base}}$ 90%	72.22	3.80	886.13	881.75	10.65
oBERTa$_{\text{base}}$ 90% Quantized	140.14	7.37	456.67	453.59	11.03
oBERTa$_{\text{base}}$ 95%	88.35	4.64	724.41	720.43	10.85
oBERTa$_{\text{base}}$ 95% Quantized	162.76	8.56	393.21	390.45	12.15
oBERTa$_{\text{MEDIUM}}$	37.94	1.99	1686.85	1685.03	8.09
oBERTa$_{\text{MEDIUM}}$ Quantized	160.48	8.44	398.80	396.47	9.27
oBERTa$_{\text{MEDIUM}}$ 90%	130.02	6.84	492.22	486.90	9.64
oBERTa$_{\text{MEDIUM}}$ 90% Quantized	259.51	13.64	246.61	244.54	7.13
oBERTa$_{\text{SMALL}}$	75.81	3.99	844.15	841.30	8.72
oBERTa$_{\text{SMALL}}$ Quantized	267.70	14.07	239.06	237.86	7.02
oBERTa$_{\text{SMALL}}$ 90%	278.93	14.67	229.43	228.41	3.43
oBERTa$_{\text{SMALL}}$ 90% Quantized	455.71	23.96	140.43	139.81	5.40

Table A.33: Inference performance of the oBERTa model family using a batch size of 64, 24 cores, and a sequence length of 384

of compression on the full dataset.

A second limitation of our work is the high computational demand required for creating public domain sparse language models. Despite amortizing the cost of compression to a few pretraining training regimes, the reduction of other language models like ALBERT [54] or XLM-R [227] require completely new training, pruning, and transfer experiments.

model	Throughput (items/sec)	Speedup	Latency Mean (ms/batch)	Latency Median (ms/batch	Latency Std (ms/batch)
oBERTa$_{\text{base}}$	4.89	1.00	204.65	204.93	1.82
oBERTa$_{\text{base}}$ Quantized	20.01	4.09	49.95	49.88	0.66
oBERTa$_{\text{base}}$ 90%	17.60	3.60	56.82	56.70	0.72
oBERTa$_{\text{base}}$ 90% Quantized	37.50	7.67	26.66	26.61	0.38
oBERTa$_{\text{base}}$ 95%	20.15	4.12	49.62	49.60	0.54
oBERTa$_{\text{base}}$ 95% Quantized	46.02	9.41	21.72	21.70	0.31
oBERTa$_{\text{MEDIUM}}$	9.59	1.96	104.28	104.33	0.90
oBERTa$_{\text{MEDIUM}}$ Quantized	41.23	8.43	24.25	24.18	0.33
oBERTa$_{\text{MEDIUM}}$ 90%	38.30	7.83	26.10	26.05	0.41
oBERTa$_{\text{MEDIUM}}$ 90% Quantized	73.28	14.99	13.64	13.60	0.19
oBERTa$_{\text{SMALL}}$	19.31	3.95	51.78	51.74	0.35
oBERTa$_{\text{SMALL}}$ Quantized	75.81	15.50	13.18	13.18	0.19
oBERTa$_{\text{SMALL}}$ 90%	68.70	14.05	14.55	14.50	0.20
oBERTa$_{\text{SMALL}}$ 90% Quantized	145.57	29.77	6.86	6.86	0.11

Table A.34: Inference performance of the oBERTa model family using a batch size of 1, 4 cores, and a sequence length of 384

model	Throughput (items/sec)	Speedup	Latency Mean (ms/batch)	Latency Median (ms/batch	Latency Std (ms/batch)
oBERTa$_{\text{base}}$	5.14	1.00	3113.07	3113.92	19.89
oBERTa$_{\text{base}}$ Quantized	22.14	4.31	722.72	719.24	11.40
oBERTa$_{\text{base}}$ 90%	17.15	3.34	932.97	931.21	5.76
oBERTa$_{\text{base}}$ 90% Quantized	39.03	7.59	409.90	408.71	4.64
oBERTa$_{\text{base}}$ 95%	19.80	3.85	808.16	806.80	4.15
oBERTa$_{\text{base}}$ 95% Quantized	46.54	9.06	343.75	342.75	4.12
oBERTa$_{\text{MEDIUM}}$	10.24	1.99	1563.00	1557.90	16.53
oBERTa$_{\text{MEDIUM}}$ Quantized	42.82	8.33	373.61	372.88	4.05
oBERTa$_{\text{MEDIUM}}$ 90%	33.69	6.56	474.88	474.25	3.64
oBERTa$_{\text{MEDIUM}}$ 90% Quantized	76.10	14.81	210.24	209.41	2.45
oBERTa$_{\text{SMALL}}$	20.41	3.97	783.81	782.99	6.59
oBERTa$_{\text{SMALL}}$ Quantized	79.57	15.48	201.07	200.60	2.12
oBERTa$_{\text{SMALL}}$ 90%	72.92	14.19	219.40	218.84	2.53
oBERTa$_{\text{SMALL}}$ 90% Quantized	139.50	27.14	114.68	114.45	1.53

Table A.35: Inference performance of the oBERTa model family using a batch size of 16, 4 cores, and a sequence length of 384

model	Throughput (items/sec)	Speedup	Latency Mean (ms/batch)	Latency Median (ms/batch	Latency Std (ms/batch)
oBERTa$_{\text{base}}$	5.06	1.00	12655.34	12680.81	57.78
oBERTa$_{\text{base}}$ Quantized	21.88	4.32	2924.89	2921.95	31.78
oBERTa$_{\text{base}}$ 90%	17.18	3.40	3724.72	3724.23	15.27
oBERTa$_{\text{base}}$ 90% Quantized	37.44	7.40	1709.44	1699.64	26.97
oBERTa$_{\text{base}}$ 95%	22.13	4.37	2892.15	2893.08	22.94
oBERTa$_{\text{base}}$ 95% Quantized	43.94	8.68	1456.53	1451.76	20.45
oBERTa$_{\text{MEDIUM}}$	10.21	2.02	1567.70	1562.90	14.53
oBERTa$_{\text{MEDIUM}}$ Quantized	42.74	8.45	374.35	373.15	4.00
oBERTa$_{\text{MEDIUM}}$ 90%	33.99	6.72	470.67	469.99	3.58
oBERTa$_{\text{MEDIUM}}$ 90% Quantized	75.64	14.95	211.53	210.80	2.61
oBERTa$_{\text{SMALL}}$	20.42	4.03	783.67	783.29	5.16
oBERTa$_{\text{SMALL}}$ Quantized	79.44	15.70	201.40	201.43	2.90
oBERTa$_{\text{SMALL}}$ 90%	71.50	14.13	223.77	223.41	1.78
oBERTa$_{\text{SMALL}}$ 90% Quantized	139.55	27.58	114.65	114.48	1.53

Table A.36: Inference performance of the oBERTa model family using a batch size of 64, 4 cores, and a sequence length of 384

A.2.13 Responsible NLP Research - Reproducibility Checklist

Datasets. We experiment with well-established benchmarks with usage in many broad domains. We do not perform any modification or augmentation in any dataset. Since datasets are not modified, we did not look for personal or sensitive content.

In our pretraining experiments, we leverage the Toronto Book Corpus (TBC) [226][61] and the

[61]https://huggingface.co/datasets/bookcorpus

Model	Throughput (items/sec)	Speedup vs BERT-Base	Speedup vs BERT-Large	Latency Mean (ms/batch)	Latency Median (ms/batch	Latency Std (ms/batch)
$\text{bert}_{\text{base}}$	4.923	1.00	5.65	203.1165	202.7077	1.3646
bert-large	0.8706	0.18	1.00	1148.6105	1145.145	9.5526
$\text{oBERTa}_{\text{base}}$	4.89	0.99	5.61	204.65	204.93	1.82
$\text{oBERTa}_{\text{base}}$ Quantized	20.01	4.07	22.99	49.95	49.88	0.66
$\text{oBERTa}_{\text{base}}$ 90%	17.60	3.57	20.21	56.82	56.70	0.72
$\text{oBERTa}_{\text{base}}$ 90% Quantized	37.50	7.62	43.07	26.66	26.61	0.38
$\text{oBERTa}_{\text{base}}$ 95%	20.15	4.09	23.14	49.62	49.60	0.54
$\text{oBERTa}_{\text{base}}$ 95% Quantized	46.02	9.35	52.86	21.72	21.70	0.31
$\text{oBERTa}_{\text{MEDIUM}}$	9.59	1.95	11.01	104.28	104.33	0.90
$\text{oBERTa}_{\text{MEDIUM}}$ Quantized	41.23	8.37	47.36	24.25	24.18	0.33
$\text{oBERTa}_{\text{MEDIUM}}$ 90%	38.30	7.78	43.99	26.10	26.05	0.41
$\text{oBERTa}_{\text{MEDIUM}}$ 90% Quantized	73.28	14.89	84.18	13.64	13.60	0.19
$\text{oBERTa}_{\text{SMALL}}$	19.31	3.92	22.18	51.78	51.74	0.35
$\text{oBERTa}_{\text{SMALL}}$ Quantized	75.81	15.40	87.07	13.18	13.18	0.19
$\text{oBERTa}_{\text{SMALL}}$ 90%	68.70	13.95	78.91	14.55	14.50	0.20
$\text{oBERTa}_{\text{SMALL}}$ 90% Quantized	145.57	29.57	167.21	6.86	6.86	0.11
pruneOFA-large 80% Quantized	12.7315	2.59	14.62	78.5322	78.3961	0.4826
prunedOFA-large 90% Quantized	11.7265	2.38	13.47	85.2647	85.1616	0.4292
obert-large	0.876	0.18	1.01	1141.5707	1138.5756	9.0121
obert-large 95%	7.508	1.53	8.62	133.1785	132.9672	1.0091
obert-large 95% Quantized	16.8077	3.41	19.31	59.4828	59.322	0.6445
pruneBERT	17.60	3.57	20.21	56.82	56.70	0.72
obert-large 97%	8.0414	1.63	9.24	124.3431	124.1421	1.0249
obert-large 97% Quantized	15.8631	3.22	18.22	63.0278	62.9979	0.6018
$\text{obert}_{\text{base}}$ 90%	18.2881	3.71	21.01	54.6688	54.5896	0.5476
$\text{obert}_{\text{base}}$ 90% Quantized	34.2797	6.96	39.37	29.1616	29.0977	0.3156
$\text{obert}_{\text{base}}$ 95%	25.1818	5.12	28.92	39.6997	39.5986	0.5805
$\text{obert}_{\text{base}}$ 95% Quantized	40.6387	8.25	46.68	24.5986	24.5222	0.3231

Table A.37: Inference performance of the other sparse models using a batch size of 1, 4 cores, and a sequence length of 384 comparing the oBERTa models to previous sparse language models such as pruneOFA [137] PruneBERT [66] and oBERT [138]

Wikipedia [143][62]. For fine-tuning we make use of SQuAD v1.1 [7] [63], SQuAD v2.0 [160] [64], Quora Duplicate Question Dataset (QQP) [65], and Multi-Genre Natural Language Inference (MNLI) [127] [66], Large Movie Review Dataset (IMDB) [163][67], Stanford Sentiment Treebank (SST-2) [161][68], and the shared task of CoNLL-2003 concerns language-independent named entity recognition (CONLL-2003) [164][69]datasets.

Models. The model used as a starting point for all our experiments is RoBERta, publicly available via HuggingFace Hub [70]. All other models presented in this paper will be released in openly-available repositories along with their compression recipes, training metrics, and hyper-parameters.

A.2.14 Computational Experiments

Upstream. During upstream pruning, due to the large size of language models and their associated teachers, we leverage 4x A100 40GB NVIDIA GPUs. We train for five epochs; an

[62]https://huggingface.co/datasets/wikipedia
[63]https://huggingface.co/datasets/squad
[64]https://huggingface.co/datasets/squadv2
[65]https://huggingface.co/datasets/glue
[66]https://huggingface.co/datasets/glue
[67]https://huggingface.co/datasets/imdb
[68]https://huggingface.co/datasets/glue
[69]https://huggingface.co/datasets/conll2003
[70]https://huggingface.co/bert-base-uncased

		Vs. BERT-Base		Vs. BERT-Large	
Model	F1	Recovery	Speed up	Recovery	Speed up
$\text{BERT}_{\text{base}}$	88.55	100.00%	1.00	97.74%	5.65
BERT-large	90.60	102.32%	0.18	100.00%	1.00
$\text{oBERTa}_{\text{base}}$	92.20	104.12%	0.99	101.77%	5.61
$\text{oBERTa}_{\text{base}}$ Quantized	93.18	105.23%	4.07	102.85%	22.99
$\text{oBERTa}_{\text{base}}$ 90%	91.00	102.77%	3.57	100.44%	20.21
$\text{oBERTa}_{\text{base}}$ 90% Quantized	89.46	101.03%	7.62	98.74%	43.07
$\text{oBERTa}_{\text{base}}$ 95%	89.84	101.46%	4.09	99.16%	23.14
$\text{oBERTa}_{\text{base}}$ 95% Quantized	88.40	99.83%	9.35	97.57%	52.86
$\text{oBERTa}_{\text{MEDIUM}}$	90.36	102.04%	1.95	99.74%	11.01
$\text{oBERTa}_{\text{MEDIUM}}$ Quantized	90.37	102.06%	8.37	99.75%	47.36
$\text{oBERTa}_{\text{MEDIUM}}$ 90%	89.26	100.80%	7.78	98.52%	43.99
$\text{oBERTa}_{\text{MEDIUM}}$ 90% Quantized	86.93	98.17%	14.89	95.95%	84.18
$\text{oBERTa}_{\text{SMALL}}$	84.87	95.84%	3.92	93.68%	22.18
$\text{oBERTa}_{\text{SMALL}}$ Quantized	84.82	95.79%	15.40	93.62%	87.07
$\text{oBERTa}_{\text{SMALL}}$ 90%	84.66	95.61%	13.95	93.45%	78.91
$\text{oBERTa}_{\text{SMALL}}$ 90% Quantized	78.71	88.89%	29.57	86.88%	167.21
pruneOFA-large 80% Quantized	90.30	101.98%	2.59	99.67%	14.62
pruneOFA-large 90% Quantized	89.96	101.59%	2.38	99.29%	13.47
oBERT-large 95%	90.19	101.85%	1.53	99.55%	1.01
oBERT-large 95% Quantized	90.21	101.87%	3.41	99.57%	8.62
pruneBERT	84.90	95.88%	3.41	93.71%	19.31
oBERT-large 97%	90.18	101.84%	13.05	99.54%	73.82
oBERT-large 97% Quantized	90.13	101.78%	1.63	99.48%	9.24
$\text{oBERT}_{\text{base}}$ 90%	88.47	99.91%	3.22	97.65%	18.22
$\text{oBERT}_{\text{base}}$ 90% Quantized	88.00	99.38%	3.71	97.13%	21.01
$\text{oBERT}_{\text{base}}$ 95%	88.19	99.59%	6.96	97.34%	39.37
$\text{oBERT}_{\text{base}}$ 95% Quantized	88.11	99.50%	5.12	97.25%	28.92

Table A.38: Speedups of the oBERTa-family compared to existing published sparse models compared to the performance of $\text{BERT}_{\text{base}}$ and BERT-large. Speedup measures the reduction in latency of MS/batch.

entire training and pruning run takes approximately 72 hours. Since the cost of such a large compute instance is high, these experiments were only run with a single seed and without major hyper-parameter exploration.

Sparse-Transfer Our experimentation on finetuning our compressed models uses the workhorse 16GB V100. Our sparse-transfer datasets vary significantly in size, and as a result, so do experiments. Finetuning for CONL2003 takes less than 10 minutes, while larger datasets like QQP take about 24 hours. Due to the number of datasets we evaluate and the number of models in the oBERTa family, we only perform experimentation with a single fixed seed.

DeepSparse inference. We pair our compressed models with DeepSparse [119], a publicly-available sparsity-aware CPU inference engine. All models are exported using the standard ONNX[71] format. For our competitive benchmarking against existing compressed language models, we leverage the model representations shared in the SparseZoo [72]. This approach means that older models, such as oBERT, may have had less optimized ONNA exports. We believe this difference in exportation causes nearly 4x improvement in the performance of oBERTa base vs. bert-base.

A.2.15 Computational Packages

All of our experimentation is done using public libraries and datasets to ensure extensibility and reproducibility. Our investigation is done using NeuralMagic's SparseML [73], which has specialized integration with HuggingFace's Transformers [74] and Datasets [75] libraries.

[71]https://onnx.ai/
[72]https://sparsezoo.neuralmagic.com/
[73]https://github.com/neuralmagic/sparseml
[74]https://github.com/huggingface/transformers
[75]https://github.com/huggingface/datasets

APPENDIX B: ROBUST AND EFFICIENT SEMANTIC RETRIEVAL

B.1 QUICK DENSE RETRIEVERS CONSUME KALE: POST TRAINING KULLBACK LEIBLER ALIGNMENT OF EMBEDDINGS FOR ASYMMETRICAL DUAL ENCODERS

B.1.1 Asymmetrical Dense Retrieval

the impact of structural pruning with asymmetrical dense retrieval can be found in table B.1. Similar to other works studying the use of knowledge distillation found [66], the use of distillation improves performance by a non-negligible level.

$layers_q$	$layers_d$	Top 20	Impact	Top 100	Impact	Top 200	Impact
12	12	79.86%	0.00%	85.84%	0.00%	88.42%	0.00%
$6_{distilbert}$	$6_{distilbert}$	73.88%	-7.49%	84.74%	-1.29%	87.26%	-1.31%
6_{KD}	12	73.99%	-7.35%	84.32%	-1.77%	86.65%	-2.00%
6_{KD}	9	71.63%	-10.30%	83.16%	-3.12%	85.82%	-2.94%
6_{KD}	6	71.00%	-11.10%	82.35%	-4.06%	85.48%	-3.32%
6_{KD}	3	68.42%	-14.32%	80.94%	-5.71%	84.24%	-4.73%
6_{KD}	2	68.39%	-14.36%	80.58%	-6.13%	84.02%	-4.98%
6_{KD}	1	56.62%	-29.10%	72.24%	-15.84%	77.81%	-12.00%
3_{KD}	12	71.72%	-10.20%	83.21%	-3.06%	85.90%	-2.85%
3_{KD}	9	68.95%	-13.66%	81.75%	-4.77%	84.79%	-4.10%
3_{KD}	6	68.09%	-14.74%	81.52%	-5.03%	84.76%	-4.13%
3_{KD}	3	65.84%	-17.55%	79.58%	-7.29%	83.41%	-5.67%
3_{KD}	2	66.81%	-16.34%	79.50%	-7.38%	82.71%	-6.45%
3_{KD}	1	54.46%	-31.81%	71.44%	-16.77%	76.59%	-13.38%
12	6_{KD}	78.78%	-1.35%	85.84%	0.01%	87.45%	-1.10%
9	6_{KD}	77.26%	-3.26%	85.18%	-0.77%	87.34%	-1.22%
6	6_{KD}	76.45%	-4.26%	84.96%	-1.03%	87.06%	-1.53%
6_{KD}	6_{KD}	75.04%	-6.03%	85.15%	-0.80%	87.45%	-1.10%
3	6_{KD}	74.49%	-6.73%	84.24%	-1.87%	86.54%	-2.13%
3_{KD}	6_{KD}	77.01%	-3.57%	85.76%	-0.09%	87.42%	-1.13%
2	6_{KD}	74.43%	-6.80%	83.68%	-2.51%	86.32%	-2.38%
1	6_{KD}	68.09%	-14.74%	79.22%	-7.71%	83.19%	-5.92%
12	3_{KD}	76.45%	-4.26%	84.49%	-1.58%	86.70%	-1.94%
9	3_{KD}	76.12%	-4.68%	84.29%	-1.80%	86.26%	-2.44%
6	3_{KD}	75.15%	-5.89%	83.43%	-2.80%	86.45%	-2.22%
6_{KD}	3_{KD}	77.40%	-3.09%	85.37%	-0.54%	87.48%	-1.06%
3_{KD}	3_{KD}	73.32%	-8.18%	83.43%	-2.80%	86.20%	-2.51%
3	3_{KD}	71.88%	-9.99%	83.66%	-2.54%	86.37%	-2.32%
2	3_{KD}	72.22%	-9.56%	81.93%	-4.55%	85.08%	-3.77%
1	3_{KD}	67.31%	-15.71%	79.25%	-7.67%	82.77%	-6.39%

Table B.1: Impact of Structural pruning with knowledge distilled variants before fine-tuning on Retrieval Accuracy on NQ passage retrieval dataset

Parameter	Possible Values
Training Length	3,40 Epochs
Initial learning rate	1e-5, 5e-5, 5e-6
Learning rate schedule	Linear
Batch size	8,128,
Negative Passages	1,8

Table B.2: Hyperparmaters used to train bi-encoder models for retrieval

Parameter	Possible Values
Training Length	1,10,100 Epochs
Initial learning rate	5e-5, 5e-4, 5e-6
Learning rate schedule	constant
Batch size	4,64,256
Loss Temperature	1, 10

Table B.3: Hyperparmaters used by KALE for aligning the embeddings of a pruned model with its uncompressed target.

B.1.2 Dense Retrieval and KALE Hyperparameters

Our experiments focus on minimal hyperparameter optimization. For training of the dense retrievers, we use the datasets described in B.2 where the shorter training lengths and smaller batch sizes correspond to MSMARCO. In contrast, the other datasets leverage longer and larger training. For using KALE, we perform task-specific grid search using the parameters described by C.2.

B.1.3 KALE

As shown in table B.4, we explore the impact of KALE for the NQ dataset, in table B.5, we explore the impact on TriviaQA, in table B.6, we evaluate the MSARCO passage retrieval, in table B.7 we explore Scifacts, and in table B.8 we explore SQUAD. The impact of pruning and KALE is fairly consistent across datasets, but there are larger losses on some smaller datasets, such as SCIfacts and SQUAD.

B.1.4 KALE and Asymmetric Training

Building on the impact of asymmetry and KALE, we explore comparing them across various datasets as shown in B.9, B.10,B.11, B.12, B.13.

Layers	KALE	Top 20	Impact	Top 100	Impact	Top 200	Impact
12	N/A	79.86%	0.00%	85.84%	0.00%	88.42%	0.00%
9	N	68.70%	-13.97%	79.97%	-6.84%	83.55%	-5.51%
9	Y	77.40%	-3.08%	84.90%	-1.10%	87.04%	-1.56%
6	N	50.69%	-36.53%	68.20%	-20.55%	73.52%	-16.85%
6	Y	75.51%	-5.45%	83.68%	-2.52%	86.18%	-2.53%
3	N	27.34%	-65.77%	43.88%	-48.88%	51.19%	-42.11%
3	Y	72.69%	-8.98%	81.14%	-5.48%	84.76%	-4.14%
2	N	27.81%	-65.18%	46.90%	-45.36%	54.54%	-38.32%
2	Y	71.83%	-10.06%	81.94%	-4.54%	84.54%	-4.39%
1	N	4.57%	-94.28%	12.22%	-85.76%	15.87%	-82.05%
1	Y	58.86%	-26.30%	71.33%	-16.90%	75.65%	-14.44%

Table B.4: Impact of structural pruning with and without KALE on the NQ retrieval dataset

Layers	KALE	Top 20	Impact	Top 100	Impact	Top 200	Impact
12	N/A	79.43%	0.00%	85.84%	0.00%	86.63%	0.00%
9	N	71.16%	-10.41%	79.97%	-5.35%	83.13%	-4.04%
9	Y	77.46%	-2.48%	84.90%	-1.28%	85.95%	-0.78%
6	N	53.98%	-32.04%	68.20%	-18.91%	74.05%	-14.52%
6	Y	75.37%	-5.11%	83.68%	-2.38%	85.25%	-1.59%
3	N	28.99%	-63.50%	43.88%	-43.84%	55.62%	-35.80%
3	Y	73.17%	-7.88%	81.14%	-3.95%	84.04%	-2.99%
2	N	33.98%	-57.22%	46.90%	-39.29%	58.52%	-32.45%
2	Y	72.39%	-8.86%	81.94%	-4.47%	83.64%	-3.45%
1	N	3.15%	-96.03%	12.22%	-90.02%	12.49%	-85.58%
1	Y	63.04%	-20.63%	71.33%	-11.33%	79.23%	-8.54%

Table B.5: Impact of structural pruning with and without KALE on the TriviaQA retrieval dataset

B.1.5 Inference Benchmarks

Evaluation of inference on GPU can be found in B.20,B.21,B.22,B.23 ,B.24,B.25 while CPU results can be found in B.14, B.15, B.16, B.17, B.18, B.19. Comparisons showing the role of speedup vs. relative degradation in retrieval accuracy can be found in B.1, B.3, and B.2

Layers	KALE	MRR@10	Impact	Top 20	Impact	Top 100	Impact	Top 200	Impact
12	N/A	32.47%	0.00%	70.47%	0.00%	88.77%	0.00%	93.84%	0.00%
9	N	27.68%	-14.74%	62.97%	-10.65%	82.01%	-7.62%	87.62%	-6.63%
9	Y	30.38%	-6.43%	67.21%	-4.64%	86.16%	-2.94%	91.85%	-2.12%
6	N	20.86%	-35.75%	52.66%	-25.27%	72.68%	-18.12%	79.20%	-15.60%
6	Y	28.71%	-11.57%	65.44%	-7.14%	84.68%	-4.60%	90.74%	-3.30%
3	N	1.49%	-95.42%	5.10%	-92.76%	11.39%	-87.17%	15.16%	-83.85%
3	Y	26.56%	-18.19%	62.36%	-11.51%	82.11%	-7.50%	88.51%	-5.68%
2	N	3.48%	-89.28%	13.55%	-80.77%	31.46%	-64.56%	38.71%	-58.75%
2	Y	26.10%	-19.61%	61.68%	-12.48%	81.96%	-7.67%	88.41%	-5.79%
1	N	0.00%	-100.00%	0.00%	-100.00%	0.00%	-100.00%	0.00%	-100.00%
1	Y	13.16%	-59.47%	34.64%	-50.84%	54.36%	-38.77%	62.82%	-33.05%

Table B.6: Impact of structural pruning with and without KALE on the MSMARCO retrieval dataset

Layers	KALE	RR@10	Impact	recall 10	Impact	NDCG@10	Impact	Top 20	Impact	Top 100	Impact	Top 200	Impact
12	N/A	59.11%	0.00%	78.71%	0.00%	62.55%	0.00%	82.38%	0.00%	90.70%	0.00%	93.77%	0.00%
9	N	25.30%	-57.20%	39.66%	-49.61%	27.46%	-56.10%	45.43%	-44.85%	71.07%	-21.64%	79.03%	-15.72%
9	Y	59.76%	1.10%	74.86%	-4.89%	62.26%	-0.46%	79.63%	-3.34%	84.87%	-6.43%	89.90%	-4.13%
6	N	8.67%	-85.33%	15.06%	-80.87%	9.16%	-85.36%	21.75%	-73.60%	22.98%	-74.66%	30.17$	-67.83%
6	Y	54.99%	-6.97%	72.53%	-7.85%	58.22%	-6.92%	77.07%	-6.45%	85.13%	-6.14%	87.70%	-6.47%
3	N	9.00%	-84.77%	16.00%	-79.67%	9.72%	-84.46%	22.40%	-72.81%	40.80%	-55.02%	51.56%	-45.01%
3	Y	55.18%	-6.65%	77.22%	-1.89%	58.30%	-6.79%	76.73%	-6.86%	82.57%	-8.96%	86.90%	-7.33%
2	N	9.65%	-83.67%	16.93%	-78.49%	10.39%	-83.39%	24.26%	-70.55%	42.66%	-52.97%	51.49%	-45.09%
2	Y	54.45%	-7.88%	71.72%	-8.88%	57.71%	-7.74%	76.07%	-7.66%	82.57%	-8.96%	85.90%	-8.39%
1	N	0.30%	-99.49%	13.30%	-83.10%	0.49%	-99.22%	1.50%	-98.18%	3.17%	-96.50%	4.23%	-95.49%
1	Y	40.52%	-31.45%	55.25%	-29.81%	43.23%	-30.89%	59.00%	-28.38%	66.83%	-26.32%	70.22%	-25.11%

Table B.7: Impact of structural pruning with and without KALE on the SCIFACTS retrieval dataset

Layers	KALE	Top 20	Impact	Top 100	Impact	Top 200	Impact
12	N/A	63.82%	0.00%	77.16%	0.00%	81.06%	0.00%
9	N	56.16%	-12.00%	71.38%	-7.49%	76.41%	-5.74%
9	Y	58.74%	-7.96%	73.54%	-4.69%	78.51%	-3.15%
6	N	42.79%	-32.95%	59.97%	-22.28%	66.63%	-17.80%
6	Y	53.51%	-16.15%	69.87%	-9.45%	75.03%	-7.44%
3	N	18.67%	-70.75%	34.42%	-55.39%	42.02%	-48.16%
3	Y	47.62%	-25.38%	64.37%	-16.58%	69.89%	-13.78%
2	N	20.82%	-67.38%	37.01%	-52.03%	45.01%	-44.47%
2	Y	46.60%	-26.98%	63.72%	-17.42%	69.53%	-14.22%
1	N	5.30%	-91.70%	11.66%	-84.89%	15.88%	-80.41%
1	Y	34.72%	-45.60%	51.39%	-33.40%	58.01%	-28.44%

Table B.8: Impact of structural pruning with and without KALE on the SQUAD retrieval dataset

Model	Layers	KALE	MRR@10	Impact	Top 20	Impact	Top 100
BERT-base	12	N	32.47%	0.00%	70.47%	0.00%	88.77%
BERT-base	6	Y	28.71%	-11.57%	65.44%	-7.14%	84.68%
$6_{kd} - 6_{kd}$	6	N	32.21%	-0.78%	69.94%	-0.75%	88.19%
$6_{db} - 6_{db}$	6	N	32.13%	-1.02%	70.37%	-0.14%	88.35%
$6_{kd} - 3_{kd}$	6	N	30.44%	-6.24%	67.82%	-3.76%	86.50%
BERT-base	3	Y	26.56%	-18.19%	62.36%	-11.51%	82.11%
$3_{kd} - 3_{kd}$	3	N	30.01%	-7.56%	67.42%	-4.33%	86.13%
$3_{kd} - 6_{kd}$	3	N	29.60%	-8.82%	66.53%	-5.59%	84.79%
$6_{kd} - 3_{kd}$	3	Y	28.19%	-13.16%	64.00%	-9.19%	82.95%
$6_{kd} - 6_{kd}$	3	Y	30.40%	-6.37%	67.62%	-4.05%	86.75%
BERT-base	2	Y	26.10%	-19.61%	61.68%	-12.48%	81.96%
$3_{kd} - 3_{kd}$	2	Y	28.57%	-12.00%	65.67%	-6.81%	84.23%
$3_{kd} - 6_{kd}$	2	Y	29.52%	-9.09%	66.16%	-6.12%	85.57%
$6_{kd} - 3_{kd}$	2	Y	28.07%	-13.54%	64.28%	-8.78%	83.24%
$6_{kd} - 6_{kd}$	2	Y	30.00%	-7.58%	66.91%	-5.06%	85.77%
BERT-base	1	Y	10.87%	-66.53%	29.80%	-57.71%	48.05%
$3_{kd} - 3_{kd}$	1	Y	19.09%	-41.21%	47.56%	-32.51%	66.69%
$3_{kd} - 6_{kd}$	1	Y	21.74%	-33.04%	52.29%	-25.80%	72.13%
$6_{kd} - 3_{kd}$	1	Y	20.82%	-35.88%	50.92%	-27.75%	71.26%
$6_{kd} - 6_{kd}$	1	Y	20.67%	-36.33%	51.81%	-26.49%	70.70%

Table B.9: Impact of model asymmetry and use of KALE for structural pruning on the MSMARCO retrieval dataset

B.2 NOISE-ROBUST DENSE RETRIEVAL VIA CONTRASTIVE ALIGNMENT POST TRAINING

B.2.1 Training HyperParameters

Detailed hyperparameters for bi-encoder training can be found in table B.26 while the parameters used for post-training alignment can be found in table B.27

B.2.2 Unaltered Bi-encoder With Noise

Summary results on the impact of noise on bi-encoder retrieval accuracy can be found in B.28 while full detailed results can be found in tables B.30, B.31, B.32, B.36, B.37, B.38, B.38, B.39, B.33, B.34,B.35.

Model	Layers	KALE	recall 20	Impact	recall 100	Impact	recall 200
BERT-base	12	N	79.86%	0.00%	85.84%	0.00%	88.42%
BERT-base	6	Y	75.51%	-5.45%	83.68%	-2.52%	86.18%
$6_{kd} - 6_{kd}$	6	N	75.04%	-6.03%	85.15%	-0.80%	87.45%
$6_{db} - 6_{db}$	6	N	73.88%	-7.49%	84.74%	-1.29%	87.26%
$6_{kd} - 3_{kd}$	6	N	77.40%	-3.09%	85.37%	-0.54%	87.48%
BERT-base	3	Y	72.69%	-8.98%	81.14%	-5.48%	84.76%
$3_{kd} - 3_{kd}$	3	N	71.88%	-9.99%	83.66%	-2.54%	86.37%
$3_{kd} - 6_{kd}$	3	N	77.01%	-3.57%	85.76%	-0.09%	87.42%
$6_{kd} - 3_{kd}$	3	Y	74.16%	-7.14%	83.43%	-2.81%	85.62%
$6_{kd} - 6_{kd}$	3	Y	69.28%	-13.25%	80.78%	-5.89%	84.10%
BERT-base	2	Y	71.83%	-10.06%	81.94%	-4.54%	84.54%
$3_{kd} - 3_{kd}$	2	Y	70.08%	-12.25%	82.71%	-3.65%	85.60%
$3_{kd} - 6_{kd}$	2	Y	75.40%	-5.58%	84.27%	-1.83%	86.81%
$6_{kd} - 3_{kd}$	2	Y	73.49%	-7.98%	83.02%	-3.29%	85.76%
$6_{kd} - 6_{kd}$	2	Y	68.42%	-14.33%	80.39%	-6.35%	83.57%
BERT-base	1	Y	58.86%	-26.30%	71.33%	-16.90%	75.65%
$3_{kd} - 3_{kd}$	1	Y	62.69%	-21.50%	77.17%	-10.10%	81.33%
$3_{kd} - 6_{kd}$	1	Y	68.14%	-14.68%	79.81%	-7.02%	82.94%
$6_{kd} - 3_{kd}$	1	Y	63.82%	-20.09%	76.57%	-10.80%	80.33%
$6_{kd} - 6_{kd}$	1	Y	60.03%	-24.83%	74.71%	-12.97%	78.64%

Table B.10: Impact of model asymmetry and use of KALE for structural pruning on the NQ retrieval dataset

Model	Layers	KALE	recall 20	Impact	recall 100	Impact	recall 200
BERT-base	12	N	79.43%	0.00%	85.03%	0.00%	86.63%
BERT-base	6	Y	75.37%	-5.11%	83.01%	-2.38%	85.25%
$6_{kd} - 6_{kd}$	6	N	79.44%	0.01%	84.96%	-0.08%	86.60%
$6_{db} - 6_{db}$	6	N	78.96%	-0.59%	84.83%	-0.23%	86.61%
$6_{kd} - 3_{kd}$	6	N	77.31%	-2.67%	84.04%	-1.17%	85.62%
BERT-base	3	Y	73.17%	-7.88%	81.67%	-3.95%	84.04%
$3_{kd} - 3_{kd}$	3	N	77.80%	-2.05%	84.11%	-1.09%	85.96%
$3_{kd} - 6_{kd}$	3	N	77.52%	-2.40%	83.91%	-1.31%	85.72%
$6_{kd} - 3_{kd}$	3	Y	74.98%	-5.60%	82.33%	-3.18%	84.35%
$6_{kd} - 6_{kd}$	3	Y	76.76%	-3.36%	83.48%	-1.82%	85.40%
BERT-base	2	Y	72.39%	-8.86%	81.23%	-4.47%	83.64%
$3_{kd} - 3_{kd}$	2	Y	76.48%	-3.71%	83.02%	-2.36%	85.16%
$3_{kd} - 6_{kd}$	2	Y	75.98%	-4.34%	82.90%	-2.50%	85.00%
$6_{kd} - 3_{kd}$	2	Y	74.60%	-6.08%	82.13%	-3.41%	84.44%
$6_{kd} - 6_{kd}$	2	Y	76.56%	-3.61%	83.32%	-2.01%	85.49%
BERT-base	1	Y	63.04%	-20.63%	75.40%	-11.33%	79.23%
$3_{kd} - 3_{kd}$	1	Y	71.66%	-9.78%	80.82%	-4.95%	83.56%
$3_{kd} - 6_{kd}$	1	Y	71.13%	-10.45%	80.23%	-5.65%	82.86%
$6_{kd} - 3_{kd}$	1	Y	68.11%	-14.25%	78.65%	-7.50%	81.89%
$6_{kd} - 6_{kd}$	1	Y	70.91%	-10.73%	80.31%	-5.55%	83.05%

Table B.11: Impact of model asymmetry and use of KALE for structural pruning on the TriviaQA retrieval dataset

Model	Layers	KALE	recall 20	Impact	recall 100	Impact	recall 200
BERT-base	12	N	63.82%	0.00%	77.16%	0.00%	81.06%
BERT-base	6	Y	53.51%	-16.15%	69.87%	-9.45%	75.03%
$6_{kd} - 6_{kd}$	6	N	54.80%	-14.14%	71.94%	-6.77%	77.73%
$6_{db} - 6_{db}$	6	N	54.60%	-14.45%	71.69%	-7.08%	77.23%
$6_{kd} - 3_{kd}$	6	N	52.97%	-17.00%	70.89%	-8.13%	76.68%
BERT-base	3	Y	47.62%	-25.38%	64.37%	-16.58%	69.89%
$3_{kd} - 3_{kd}$	3	N	55.05%	-13.74%	71.98%	-6.72%	77.76%
$3_{kd} - 6_{kd}$	3	N	48.86%	-23.43%	67.85%	-12.06%	74.04%
$6_{kd} - 3_{kd}$	3	Y	44.65%	-30.04%	63.77%	-17.35%	70.79%
$6_{kd} - 6_{kd}$	3	Y	45.36%	-28.93%	64.14%	-16.87%	71.07%
BERT-base	2	Y	48.43%	-24.11%	67.02%	-13.14%	73.19%
$3_{kd} - 3_{kd}$	2	Y	48.43%	-24.11%	67.02%	-13.14%	73.19%
$3_{kd} - 6_{kd}$	2	Y	43.45%	-31.92%	62.75%	-18.68%	69.74%
$6_{kd} - 3_{kd}$	2	Y	42.90%	-32.78%	62.52%	-18.97%	69.47%
$6_{kd} - 6_{kd}$	2	Y	35.08%	-45.03%	52.74%	-31.65%	59.93%
BERT-base	1	Y	34.72%	-45.60%	51.39%	-33.40%	58.01%
$3_{kd} - 3_{kd}$	1	Y	36.19%	-43.29%	55.62%	-27.92%	62.92%
$3_{kd} - 6_{kd}$	1	Y	34.75%	-45.55%	52.26%	-32.27%	59.35%
$6_{kd} - 3_{kd}$	1	Y	32.18%	-49.58%	50.88%	-34.06%	58.52%
$6_{kd} - 6_{kd}$	1	Y	35.08%	-45.03%	52.74%	-31.65%	59.93%

Table B.12: Impact of model asymmetry and use of KALE for structural pruning on the SQUAD retrieval dataset

Model	Layers	KALE	recip_rank	Impact	NDC@10	Impact	Recall 20
BERT-base	12	N	59.11%	0.00%	62.55%	0.00%	82.38%
BERT-base	6	Y	54.99%	-6.97%	58.22%	-6.92%	77.07%
$6_{kd} - 6_{kd}$	6	N	65.52%	10.84%	67.87%	8.51%	83.92%
$6_{db} - 6_{db}$	6	N	66.25%	12.08%	67.81%	8.41%	82.16%
$6_{kd} - 3_{kd}$	6	N	61.90%	4.72%	65.30%	4.40%	82.48%
BERT-base	3	Y	55.18%	-6.65%	58.30%	-6.79%	76.73%
$3_{kd} - 3_{kd}$	3	N	65.32%	10.51%	67.51%	7.93%	84.36%
$3_{kd} - 6_{kd}$	3	N	62.78%	6.21%	64.86%	3.69%	79.80%
$6_{kd} - 3_{kd}$	3	Y	62.07%	5.01%	64.73%	3.49%	82.57%
$6_{kd} - 6_{kd}$	3	Y	61.82%	4.58%	65.41%	4.57%	82.41%
BERT-base	2	Y	54.45%	-7.88%	57.71%	-7.74%	76.07%
$3_{kd} - 3_{kd}$	2	Y	61.78%	4.52%	64.78%	3.57%	82.76%
$3_{kd} - 6_{kd}$	2	Y	61.41%	3.89%	63.61%	1.69%	82.46%
$6_{kd} - 3_{kd}$	2	Y	61.82%	4.58%	64.80%	3.60%	82.51%
$6_{kd} - 6_{kd}$	2	Y	62.09%	5.04%	65.27%	4.35%	81.51%
BERT-base	1	Y	40.52%	-31.45%	43.23%	-30.89%	59.00%
$3_{kd} - 3_{kd}$	1	Y	42.93%	-27.37%	44.19%	-29.35%	61.06%
$3_{kd} - 6_{kd}$	1	Y	42.33%	-28.39%	44.03%	-29.61%	63.33%
$6_{kd} - 3_{kd}$	1	Y	42.72%	-27.73%	45.68%	-26.97%	65.81%
$6_{kd} - 6_{kd}$	1	Y	45.60%	-22.86%	48.83%	-21.93%	69.11%

Table B.13: Impact of model asymmetry and use of KALE for structural pruning on the SCIFACTS retrieval dataset

	items/sec	Full Time	Mean Time	95th	50th	5th	99th
Run 1	44.890	80.414	2.17E-02	2.92E-02	2.09E-02	1.97E-02	3.07E-02
Run 2	48.370	74.628	2.01E-02	2.11E-02	2.00E-02	1.96E-02	2.22E-02
Run 3	47.290	76.334	2.06E-02	2.19E-02	2.04E-02	1.96E-02	2.28E-02
Run 4	48.260	74.810	2.01E-02	2.13E-02	2.00E-02	1.95E-02	2.22E-02
Run 5	47.580	75.872	2.04E-02	2.14E-02	2.03E-02	1.98E-02	2.28E-02
average	47.278	76.412	2.06E-02	2.30E-02	2.03E-02	1.96E-02	2.41E-02
stdev	1.410	2.348	6.46E-04	3.49E-03	3.65E-04	1.04E-04	3.68E-03
CI	1.236	2.058	5.66E-04	3.06E-03	3.20E-04	9.14E-05	3.23E-03
Lower	46.042	74.353	2.00E-02	1.99E-02	2.00E-02	1.96E-02	2.09E-02
High	48.514	78.470	2.12E-02	2.60E-02	2.06E-02	1.97E-02	2.74E-02

Table B.14: Inference Benchmark for 12-layer Query encoder on a CPU using ONNX

	items/sec	Full Time	Mean Time	95th	50th	5th	99th
Run 1	63.200	57.808	1.54E-02	1.65E-02	1.52E-02	1.49E-02	2.20E-02
Run 2	63.570	56.787	1.52E-02	1.60E-02	1.50E-02	1.48E-02	1.70E-02
Run 3	62.740	57.537	1.54E-02	1.64E-02	1.52E-02	1.48E-02	1.76E-02
Run 4	63.440	56.908	1.52E-02	1.59E-02	1.51E-02	1.48E-02	1.70E-02
Run 5	63.250	57.077	1.53E-02	1.60E-02	1.51E-02	1.48E-02	1.69E-02
average	63.240	57.223	1.53E-02	1.62E-02	1.51E-02	1.48E-02	1.81E-02
stdev	0.316	0.433	1.16E-04	2.49E-04	6.48E-05	6.69E-05	2.20E-03
CI	0.277	0.380	1.02E-04	2.18E-04	5.68E-05	5.86E-05	1.93E-03
Lower	62.963	56.844	1.52E-02	1.59E-02	1.51E-02	1.48E-02	1.62E-02
High	63.517	57.603	1.54E-02	1.64E-02	1.52E-02	1.49E-02	2.00E-02

Table B.15: Inference Benchmark for 9-layer Query encoder on a CPU using ONNX

	items/sec	Full Time	Mean Time	95th	50th	5th	99th
Run 1	91.090	39.631	1.04E-02	1.11E-02	1.03E-02	1.02E-02	1.19E-02
Run 2	90.990	39.677	1.04E-02	1.11E-02	1.03E-02	1.01E-02	1.22E-02
Run 3	91.290	39.547	1.04E-02	1.11E-02	1.03E-02	1.01E-02	1.22E-02
Run 4	89.420	40.372	1.06E-02	1.24E-02	1.02E-02	1.01E-02	1.51E-02
Run 5	89.140	40.499	1.07E-02	1.21E-02	1.03E-02	1.01E-02	1.49E-02
average	90.386	39.945	1.05E-02	1.16E-02	1.03E-02	1.01E-02	1.32E-02
stdev	1.020	0.452	1.23E-04	6.03E-04	3.95E-05	4.27E-05	1.61E-03
CI	0.894	0.396	1.08E-04	5.29E-04	3.47E-05	3.74E-05	1.41E-03
Lower	89.492	39.549	1.04E-02	1.10E-02	1.03E-02	1.01E-02	1.18E-02
High	91.280	40.342	1.06E-02	1.21E-02	1.03E-02	1.02E-02	1.47E-02

Table B.16: Inference Benchmark for 6-layer Query encoder on a CPU using ONNX

	items/sec	Full Time	Mean Time	95th	50th	5th	99th
Run 1	166.340	21.704	5.47E-03	5.84E-03	5.40E-03	5.35E-03	6.34E-03
Run 2	164.830	21.902	5.53E-03	6.14E-03	5.40E-03	5.31E-03	7.35E-03
Run 3	167.570	21.544	5.43E-03	5.87E-03	5.34E-03	5.30E-03	6.42E-03
Run 4	165.370	21.830	5.51E-03	6.11E-03	5.39E-03	5.30E-03	6.96E-03
Run 5	165.950	21.755	5.49E-03	5.92E-03	5.40E-03	5.32E-03	6.54E-03
average	166.012	21.747	5.49E-03	5.98E-03	5.39E-03	5.32E-03	6.72E-03
stdev	1.043	0.136	3.58E-05	1.41E-04	2.49E-05	2.20E-05	4.23E-04
CI	0.914	0.119	3.14E-05	1.23E-04	2.18E-05	1.93E-05	3.71E-04
Lower	165.098	21.628	5.45E-03	5.86E-03	5.37E-03	5.30E-03	6.35E-03
High	166.926	21.867	5.52E-03	6.10E-03	5.41E-03	5.33E-03	7.09E-03
BERT-base	2	Y	54.45%	-7.88%	57.71%	-7.74%	76.07%

Table B.17: Inference Benchmark for 3-layer Query encoder on a CPU using ONNX

	items/sec	Full Time	Mean Time	95th	50th	5th	99th
Run 1	228.690	15.786	3.85E-03	4.53E-03	3.72E-03	3.67E-03	5.29E-03
Run 2	230.420	15.668	3.81E-03	4.24E-03	3.74E-03	3.65E-03	4.72E-03
Run 3	228.800	15.779	3.84E-03	4.23E-03	3.77E-03	3.73E-03	4.68E-03
Run 4	230.530	15.661	3.81E-03	4.23E-03	3.74E-03	3.68E-03	4.63E-03
Run 5	229.890	15.704	3.82E-03	4.25E-03	3.75E-03	3.70E-03	4.64E-03
average	229.666	15.720	3.83E-03	4.29E-03	3.74E-03	3.69E-03	4.79E-03
stdev	0.876	0.060	1.72E-05	1.32E-04	1.84E-05	3.00E-05	2.81E-04
CI	0.768	0.053	1.51E-05	1.16E-04	1.61E-05	2.63E-05	2.47E-04
Lower	228.898	15.667	3.81E-03	4.18E-03	3.73E-03	3.66E-03	4.55E-03
High	230.434	15.772	3.84E-03	4.41E-03	3.76E-03	3.71E-03	5.04E-03

Table B.18: Inference Benchmark for two-layer Query encoder on a CPU using ONNX

	items/sec	Full Time	Mean Time	95th	50th	5th	99th
Run 1	378.680	9.534	2.14E-03	2.39E-03	2.10E-03	2.08E-03	2.88E-03
Run 2	378.950	9.528	2.14E-03	2.31E-03	2.11E-03	2.08E-03	2.66E-03
Run 3	377.750	9.558	2.13E-03	2.30E-03	2.12E-03	2.06E-03	2.67E-03
Run 4	376.560	9.588	2.16E-03	2.35E-03	2.12E-03	2.06E-03	2.74E-03
Run 5	380.730	9.483	2.14E-03	2.30E-03	2.11E-03	2.08E-03	2.66E-03
average	378.534	9.538	2.15E-03	2.33E-03	2.11E-03	2.07E-03	2.72E-03
stdev	1.543	0.039	7.46E-06	3.64E-05	8.72E-06	9.49E-06	9.64E-05
CI	1.353	0.034	6.54E-06	3.19E-05	7.65E-06	8.31E-06	8.45E-05
Lower	377.181	9.504	2.14E-03	2.30E-03	2.11E-03	2.06E-03	2.64E-03
High	379.887	9.572	2.15E-03	2.36E-03	2.12E-03	2.08E-03	2.81E-03

Table B.19: Inference Benchmark for one layer Query encoder on a CPU using ONNX

	items/sec	Full Time	Mean Time	95th	50th	5th	99th
Run 1	103.16	35.00	9.22E-03	9.33E-03	9.16E-03	9.08E-03	1.20E-02
Run 2	111.51	32.36	8.50E-03	8.61E-03	8.47E-03	8.42E-03	8.73E-03
Run 3	114.02	31.66	8.31E-03	8.41E-03	8.28E-03	8.22E-03	8.60E-03
Run 4	90.39	39.94	1.06E-02	1.07E-02	1.05E-02	1.04E-02	1.25E-02
Run 5	110.18	32.77	8.62E-03	8.74E-03	8.58E-03	8.51E-03	9.06E-03
average	105.85	34.35	9.04E-03	9.15E-03	9.00E-03	8.93E-03	1.02E-02
stdev	9.54	3.37	9.17E-04	9.19E-04	9.04E-04	9.02E-04	1.92E-03
CI	8.36	2.95	8.04E-04	8.06E-04	7.92E-04	7.91E-04	1.68E-03
Lower	97.49	31.40	8.24E-03	8.35E-03	8.21E-03	8.14E-03	8.50E-03
High	114.21	37.30	9.85E-03	9.96E-03	9.79E-03	9.73E-03	1.19E-02

Table B.20: Inference Benchmark for 12-layer Query encoder on a T4 GPU

	items/sec	Full Time	Mean Time	95th	50th	5th	99th
Run 1	140.35	25.72	6.69E-03	6.78E-03	6.66E-03	6.61E-03	6.94E-03
Run 2	148.25	24.35	6.31E-03	6.52E-03	6.26E-03	6.22E-03	6.64E-03
Run 3	147.04	24.55	6.37E-03	6.47E-03	6.32E-03	6.28E-03	7.19E-03
Run 4	116.15	31.08	8.14E-03	8.25E-03	8.09E-03	8.01E-03	1.09E-02
Run 5	145.68	24.78	6.44E-03	6.50E-03	6.39E-03	6.35E-03	8.83E-03
average	139.49	26.10	6.79E-03	6.91E-03	6.74E-03	6.69E-03	8.11E-03
stdev	13.39	2.84	7.70E-04	7.62E-04	7.66E-04	7.52E-04	1.79E-03
CI	11.74	2.49	6.75E-04	6.68E-04	6.72E-04	6.59E-04	1.57E-03
Lower	127.75	23.61	6.11E-03	6.24E-03	6.07E-03	6.04E-03	6.54E-03
High	151.23	28.58	7.46E-03	7.57E-03	7.42E-03	7.35E-03	9.67E-03

Table B.21: Inference Benchmark for 9-layer Query encoder on a T4 GPU

	items/sec	Full Time	Mean Time	95th	50th	5th	99th
Run 1	163.72	22.05	5.67E-03	5.75E-03	5.62E-03	5.56E-03	7.75E-03
Run 2	161.90	22.30	5.74E-03	5.81E-03	5.70E-03	5.63E-03	6.17E-03
Run 3	165.07	21.87	5.62E-03	5.70E-03	5.58E-03	5.51E-03	6.86E-03
Run 4	189.71	19.03	4.84E-03	4.92E-03	4.82E-03	4.77E-03	5.07E-03
Run 5	181.29	19.91	5.07E-03	5.92E-03	4.94E-03	4.88E-03	6.68E-03
average	172.34	21.03	5.39E-03	5.62E-03	5.33E-03	5.27E-03	6.51E-03
stdev	12.43	1.47	4.07E-04	3.99E-04	4.17E-04	4.11E-04	9.85E-04
CI	10.89	1.29	3.56E-04	3.50E-04	3.65E-04	3.61E-04	8.63E-04
Lower	161.44	19.75	5.03E-03	5.27E-03	4.97E-03	4.91E-03	5.64E-03
High	183.23	22.32	5.74E-03	5.97E-03	5.70E-03	5.63E-03	7.37E-03

Table B.22: Inference Benchmark for 6-layer Query encoder on a T4 GPU

	items/sec	Full Time	Mean Time	95th	50th	5th	99th
Run 1	269.73	13.39	3.28E-03	3.30E-03	3.26E-03	3.20E-03	3.87E-03
Run 2	282.90	12.76	3.12E-03	3.38E-03	3.23E-03	2.65E-03	4.39E-03
Run 3	268.47	13.45	3.30E-03	3.31E-03	3.28E-03	3.25E-03	3.76E-03
Run 4	318.47	11.34	2.74E-03	2.79E-03	2.72E-03	2.69E-03	3.17E-03
Run 5	357.68	10.09	2.43E-03	2.50E-03	2.41E-03	2.39E-03	2.69E-03
average	299.45	12.21	2.97E-03	3.05E-03	2.98E-03	2.84E-03	3.58E-03
stdev	38.31	1.45	3.78E-04	3.90E-04	3.93E-04	3.75E-04	6.58E-04
CI	33.58	1.27	3.31E-04	3.42E-04	3.45E-04	3.29E-04	5.77E-04
Lower	265.87	10.93	2.64E-03	2.71E-03	2.64E-03	2.51E-03	3.00E-03
High	333.03	13.48	3.30E-03	3.40E-03	3.33E-03	3.16E-03	4.16E-03

Table B.23: Inference Benchmark for 3-layer Query encoder on a T4 GPU

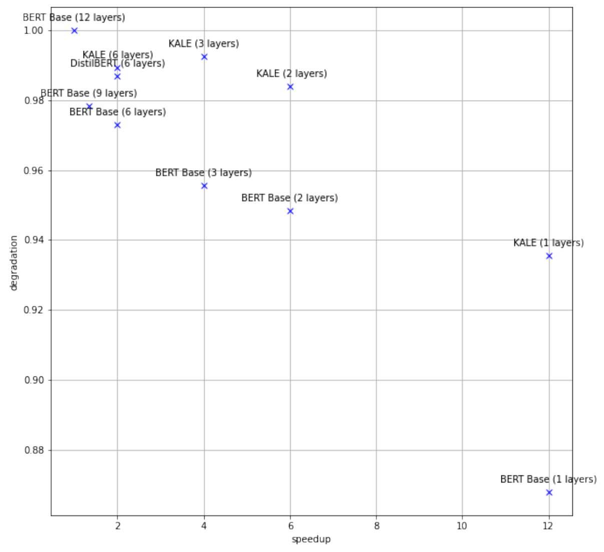

Figure B.1: Speedup vs. Retrieval accuracy of compressed models on NQ passage retrieval with a recall set 200.

	items/sec	Full Time	Mean Time	95th	50th	5th	99th
Run 1	465.83	7.75	1.78E-03	1.83E-03	1.76E-03	1.74E-03	2.53E-03
Run 2	435.46	8.29	1.92E-03	2.01E-03	1.91E-03	1.89E-03	2.04E-03
Run 3	471.01	7.67	1.77E-03	1.84E-03	1.75E-03	1.74E-03	1.95E-03
Run 4	413.49	8.73	2.02E-03	2.06E-03	2.00E-03	1.96E-03	2.61E-03
Run 5	421.32	8.57	1.98E-03	2.05E-03	1.96E-03	1.94E-03	2.07E-03
average	441.42	8.20	1.89E-03	1.96E-03	1.88E-03	1.86E-03	2.24E-03
stdev	25.94	0.48	1.15E-04	1.12E-04	1.15E-04	1.07E-04	3.07E-04
CI	22.73	0.42	1.00E-04	9.83E-05	1.01E-04	9.34E-05	2.69E-04
Lower	418.69	7.78	1.79E-03	1.86E-03	1.78E-03	1.76E-03	1.97E-03
High	464.16	8.62	1.99E-03	2.05E-03	1.98E-03	1.95E-03	2.51E-03

Table B.24: Inference Benchmark for 2-layer Query encoder on a T4 GPU

	items/sec	Full Time	Mean Time	95th	50th	5th	99th
Run 1	627.64	5.75	1.22E-03	1.26E-03	1.21E-03	1.20E-03	1.28E-03
Run 2	673.96	5.36	1.13E-03	1.18E-03	1.12E-03	1.11E-03	1.22E-03
Run 3	651.45	5.54	1.18E-03	1.24E-03	1.17E-03	1.16E-03	1.28E-03
Run 4	677.99	5.33	1.12E-03	1.19E-03	1.11E-03	1.10E-03	1.22E-03
Run 5	672.16	5.37	1.13E-03	1.18E-03	1.12E-03	1.11E-03	1.22E-03
average	660.64	5.47	1.15E-03	1.21E-03	1.14E-03	1.14E-03	1.24E-03
stdev	21.12	0.18	4.28E-05	3.74E-05	4.44E-05	4.25E-05	3.30E-05
CI	18.51	0.16	3.75E-05	3.27E-05	3.89E-05	3.72E-05	2.89E-05
Lower	642.13	5.31	1.12E-03	1.18E-03	1.11E-03	1.10E-03	1.21E-03
High	679.15	5.63	1.19E-03	1.24E-03	1.18E-03	1.17E-03	1.27E-03

Table B.25: Inference Benchmark for 1-layer Query encoder on a T4 GPU

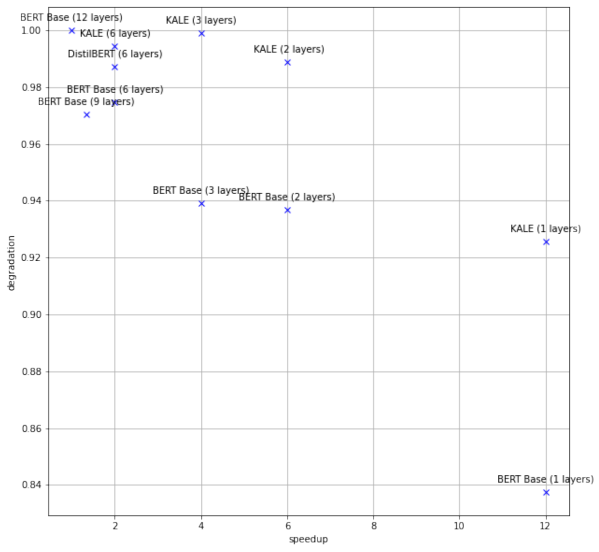

Figure B.2: Speedup vs. Retrieval accuracy of compressed models on NQ passage retrieval with a recall set of 100.

Dataset	Batch Size	Learning Rate	Train Epochs	Negative Passages
NQ	128	1e-5	40	1
TrivaQA	128	1e-5	40	1
MSMARCO	8	5e-6	3	8

Table B.26: Model Training parameters across tasks. NQ and TriviaQA are trained using 4 V100s, while MSMARCO uses a single V100

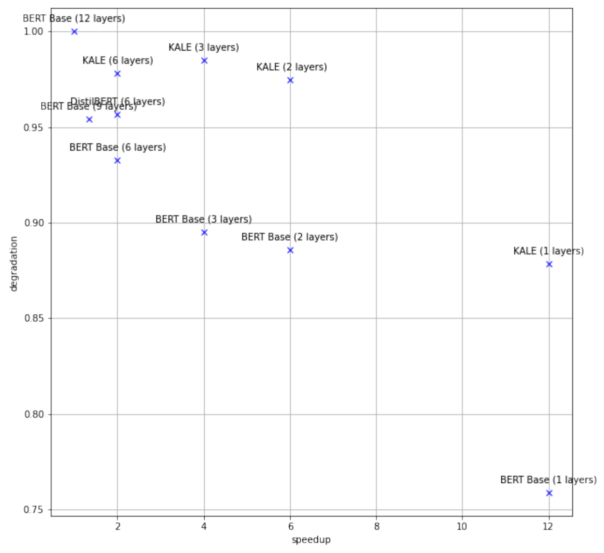

Figure B.3: Speedup vs. Retrieval accuracy of compressed models on NQ passage retrieval with a recall set of 20.

Dataset	Learning Rate	$\tau_{positive}$	$\tau_{negative}$	τ_{anchor}	$\tau_{ranking}$	Batch Size	Training Time
NQ	5.00E-05	2	0.2	2	0.7	2048	1 hour
TriviaQA	5.00E-05	1	0.1	2	1	2048	1.5 hours
MSMARCO	5.00E-05	1	0.1	2	1	2048	5 hours

Table B.27: CAPOT optimal hyperparameters across datasets. Models were generally trained for ten epochs, but we find that a single epoch can provide 95% of the final performance increase.

Dataset	20	20 w/noise	Loss	100	100 w/noise	Loss	200	200 w/noise	Loss
NQ	79.73%	72.30%	-9.31%	85.98%	81.58%	-5.12%	88.25%	84.31%	-4.47%
MS Marco	71.63%	58.45%	-18.40%	88.79%	58.68%	-33.91%	93.78%	80.63%	-14.02%
TriviaQA	79.40%	75.86%	-4.46%	85.01%	82.71%	-2.70%	86.66%	84.89%	-2.04%

Table B.28: Retrieval accuracy for Bi-encoders on unaltered and noisy queries with recall sets of 20,100, and 200 documents.

Dataset	20	20 w/noise	Loss	100	100 w/noise	Impact	200	200 w/noise	Loss
NQ	79.73%	67.83%	-14.93%	85.98%	78.71%	-8.45%	88.25%	81.84%	-7.27%
MS Marco	71.63%	41.77%	-41.69%	88.79%	70.13%	-21.02%	93.78%	82.47%	-12.06%
TriviaQA	79.40%	72.71%	-8.43%	85.01%	81.15%	-4.55%	86.66%	83.72%	-3.39%

Table B.29: Retrieval accuracy for Bi-encoders on unaltered and character-based noisy queries (typos) with recall sets of 20,100, and 200 documents

Noise	Seed 1	Seed 2	Seed 3	Seed 4	Seed 5	Median	STD	CI	Upper Bound	Lower Bound
None	79.86%	79.78%	79.47%	79.64%	79.89%	79.73%	1.72E-03	1.51E-03	79.88%	79.58%
Determiner	74.46%	74.65%	74.88%	74.54%	74.40%	74.59%	1.86E-03	1.63E-03	74.75%	74.42%
Synonym	68.50%	68.23%	68.39%	68.84%	68.37%	68.47%	2.30E-03	2.01E-03	68.67%	68.26%
Lemma	74.38%	74.10%	74.46%	74.35%	74.35%	74.33%	1.35E-03	1.18E-03	74.45%	74.21%
Stem	74.38%	74.10%	74.46%	73.93%	74.35%	74.24%	2.19E-03	1.92E-03	74.44%	74.05%
RCS	67.06%	67.37%	67.78%	66.62%	66.79%	67.12%	4.65E-03	4.08E-03	67.53%	66.72%
KCS	67.04%	67.01%	67.40%	66.34%	67.29%	67.01%	4.09E-03	3.59E-03	67.37%	66.66%
CD	69.20%	69.00%	69.97%	69.31%	69.25%	69.35%	3.68E-03	3.23E-03	69.67%	69.02%
RW	76.45%	76.40%	76.48%	76.51%	76.45%	76.46%	4.11E-04	3.60E-04	76.50%	76.42%
BT	72.35%	72.13%	71.91%	71.63%	72.11%	72.03%	2.70E-03	2.37E-03	72.26%	71.79%
Paraphrase	72.19%	72.19%	71.72%	71.52%	72.38%	72.00%	3.62E-03	3.17E-03	72.32%	71.68%
Average	72.35%	72.27%	72.45%	72.11%	72.33%	72.30%	1.24E-03	1.09E-03	72.41%	72.19%
Typos	67.77%	67.79%	68.38%	67.42%	67.77%	67.83%	4.14E-03	3.63E-03	68.19%	67.47%

Table B.30: Baseline retrieval accuracy at recall set size 20 Performance on NQ with five random seeds with and without noisy queries.

Noise	Seed 1	Seed 2	Seed 3	Seed 4	Seed 5	Median	STD	CI	Upper Bound	Lower Bound
None	85.98%	86.01%	86.15%	85.93%	85.82%	85.98%	1.21E-03	1.06E-03	86.08%	85.87%
Determiner	83.49%	83.35%	83.41%	83.57%	83.43%	83.45%	8.45E-04	7.40E-04	83.53%	83.38%
Synonym	79.61%	79.36%	79.78%	79.58%	79.75%	79.62%	1.66E-03	1.45E-03	79.76%	79.47%
Lemma	82.85%	83.21%	82.94%	82.71%	82.96%	82.94%	1.83E-03	1.60E-03	83.10%	82.78%
Stem	82.88%	83.21%	82.94%	82.71%	82.96%	82.94%	1.80E-03	1.58E-03	83.10%	82.78%
RCS	78.50%	79.00%	79.11%	77.92%	78.78%	78.66%	4.76E-03	4.17E-03	79.08%	78.25%
KCS	78.01%	77.92%	78.42%	77.73%	78.03%	78.02%	2.53E-03	2.22E-03	78.24%	77.80%
CD	79.75%	79.36%	79.53%	79.00%	79.64%	79.46%	2.92E-03	2.56E-03	79.71%	79.20%
RW	85.07%	85.07%	84.85%	84.82%	85.04%	84.97%	1.25E-03	1.09E-03	85.08%	84.86%
BT	80.14%	79.92%	80.03%	79.47%	80.11%	79.93%	2.71E-03	2.38E-03	80.17%	79.70%
Paraphrase	81.44%	81.44%	81.19%	81.44%	81.30%	81.36%	1.13E-03	9.92E-04	81.46%	81.26%
Average	81.61%	81.62%	81.67%	81.35%	81.62%	81.58%	1.25E-03	1.10E-03	81.69%	81.47%
Typos	78.75%	78.76%	79.02%	78.22%	78.82%	78.71%	0.34%	0.30%	79.01%	78.42%

Table B.31: Baseline retrieval accuracy at recall set size 100 Performance on NQ with five random seeds with and without noisy queries.

Noise	Seed 1	Seed 2	Seed 3	Seed 4	Seed 5	Median	STD	CI	Upper Bound	Lower Bound
None	88.42%	87.89%	88.23%	88.28%	88.42%	88.25%	2.16E-03	1.89E-03	88.44%	88.06%
Determiner	86.09%	85.82%	86.04%	85.87%	86.12%	85.99%	1.36E-03	1.19E-03	86.11%	85.87%
Synonym	82.85%	82.85%	82.66%	82.80%	82.96%	82.83%	1.11E-03	9.71E-04	82.92%	82.73%
Lemma	86.04%	85.37%	82.94%	85.37%	85.93%	85.13%	1.26E-02	1.11E-02	86.24%	84.02%
Stem	86.01%	85.37%	85.90%	85.37%	85.93%	85.72%	3.16E-03	2.77E-03	85.99%	85.44%
RCS	81.72%	82.08%	82.49%	81.86%	81.63%	81.96%	3.44E-03	3.02E-03	82.26%	81.65%
KCS	81.69%	81.47%	81.72%	80.97%	81.75%	81.52%	3.26E-03	2.85E-03	81.80%	81.23%
CD	79.75%	82.60%	82.83%	82.44%	82.58%	82.04%	1.29E-02	1.13E-02	83.17%	80.91%
RW	87.34%	87.40%	87.26%	87.34%	87.37%	87.34%	5.18E-04	4.54E-04	87.39%	87.30%
BT	82.52%	82.19%	82.94%	82.47%	82.60%	82.54%	2.70E-03	2.36E-03	82.78%	82.31%
Paraphrase	83.93%	83.93%	83.99%	84.27%	84.32%	84.09%	1.90E-03	1.66E-03	84.25%	83.92%
Average	84.22%	84.27%	84.27%	84.28%	84.51%	84.31%	1.15E-03	1.01E-03	84.41%	84.21%
Typos	81.05%	82.05%	82.35%	81.75%	81.99%	81.84%	6.52E-03	5.72E-03	82.41%	81.27%

Table B.32: Baseline retrieval accuracy at recall set size 200 Performance on NQ with five random seeds with and without noisy queries.

Noise	Seed 1	Seed 2	Seed 3	Seed 4	Seed 5	Median	STD	CI	Upper Bound	Lower Bound
None	79.37%	79.56%	79.58%	79.32%	79.17%	79.40%	1.71E-03	1.50E-03	79.55%	79.25%
Determiner	77.40%	77.44%	77.39%	77.54%	77.02%	77.36%	1.99E-03	1.74E-03	77.53%	77.18%
Synonym	74.93%	75.18%	75.06%	75.21%	75.15%	75.11%	1.11E-03	9.76E-04	75.20%	75.01%
Lemma	79.11%	79.11%	79.15%	78.95%	79.02%	79.06%	8.91E-04	7.81E-04	79.14%	78.98%
Stem	78.11%	78.35%	78.41%	77.98%	78.18%	78.21%	1.77E-03	1.55E-03	78.36%	78.05%
RCS	72.41%	72.94%	72.67%	72.60%	72.60%	72.64%	1.92E-03	1.68E-03	72.81%	72.48%
KCS	72.24%	72.39%	72.52%	72.34%	72.21%	72.34%	1.23E-03	1.08E-03	72.45%	72.23%
CD	73.31%	73.16%	73.09%	73.10%	73.06%	73.14%	9.77E-04	8.56E-04	73.23%	73.06%
RW	78.16%	78.18%	78.16%	77.80%	77.96%	78.05%	1.69E-03	1.48E-03	78.20%	77.90%
BT	73.76%	73.93%	73.77%	75.08%	73.69%	74.05%	5.85E-03	5.13E-03	74.56%	73.53%
Paraphrase	75.04%	75.19%	75.36%	75.08%	74.84%	75.10%	1.89E-03	1.66E-03	75.27%	74.94%
Average	75.80%	75.95%	75.92%	75.91%	75.72%	75.86%	9.75E-04	8.54E-04	75.95%	75.78%
Typos	72.65%	72.83%	72.76%	72.68%	72.62%	72.71%	1.38E-03	1.21E-03	72.83%	72.59%

Table B.33: Baseline retrieval accuracy at recall set size 20 Performance on TrivaQA with five random seeds with and without noisy queries.

Noise	Seed 1	Seed 2	Seed 3	Seed 4	Seed 5	Median	STD	CI	Upper Bound	Lower Bound
None	85.03%	84.95%	85.10%	85.10%	84.88%	85.01%	9.41E-04	8.25E-04	85.09%	84.93%
Determiner	83.78%	83.77%	84.02%	83.90%	83.70%	83.83%	1.26E-03	1.11E-03	83.95%	83.72%
Synonym	82.37%	82.41%	82.52%	82.37%	82.27%	82.39%	8.87E-04	7.78E-04	82.47%	82.31%
Lemma	84.81%	84.82%	84.82%	84.92%	84.68%	84.81%	8.51E-04	7.46E-04	84.89%	84.74%
Stem	84.40%	84.32%	84.45%	84.42%	84.20%	84.36%	1.00E-03	8.78E-04	84.45%	84.27%
RCS	80.92%	81.00%	81.00%	80.92%	80.92%	80.95%	4.13E-04	3.62E-04	80.99%	80.92%
KCS	81.12%	81.03%	81.39%	81.07%	81.19%	81.16%	1.44E-03	1.26E-03	81.29%	81.03%
CD	81.34%	81.26%	81.38%	81.35%	81.30%	81.33%	4.83E-04	4.23E-04	81.37%	81.28%
RW	84.33%	84.27%	84.36%	77.80%	84.20%	82.99%	2.91E-02	2.55E-02	85.54%	80.45%
BT	80.33%	80.08%	80.20%	82.26%	80.12%	80.60%	9.33E-03	8.18E-03	81.42%	79.78%
Paraphrase	82.46%	82.50%	82.33%	82.50%	82.21%	82.40%	1.28E-03	1.12E-03	82.51%	82.29%
Average	82.81%	82.77%	82.87%	82.42%	82.70%	82.71%	1.76E-03	1.54E-03	82.87%	82.56%
Typos	81.13%	81.10%	81.26%	81.11%	81.14%	81.15%	0.08%	0.07%	81.21%	81.08%

Table B.34: Baseline retrieval accuracy at recall set size 100 Performance on TrivaQA with five random seeds with and without noisy queries.

Noise	Seed 1	Seed 2	Seed 3	Seed 4	Seed 5	Median	STD	CI	Upper Bound	Lower Bound
None	86.87%	86.53%	86.61%	86.63%	86.67%	86.66%	1.29E-03	1.13E-03	86.77%	86.55%
Determiner	85.73%	85.66%	85.62%	85.65%	85.73%	85.68%	5.12E-04	4.48E-04	85.73%	85.64%
Synonym	84.58%	84.46%	84.46%	84.67%	84.58%	84.55%	8.99E-04	7.88E-04	84.63%	84.47%
Lemma	86.72%	86.35%	86.45%	86.44%	86.54%	86.50%	1.41E-03	1.23E-03	86.62%	86.38%
Stem	86.38%	86.04%	86.09%	86.15%	86.24%	86.18%	1.33E-03	1.17E-03	86.30%	86.06%
RCS	83.64%	83.43%	83.48%	83.55%	83.55%	83.53%	7.77E-04	6.81E-04	83.60%	83.46%
KCS	83.92%	83.72%	83.89%	83.66%	83.80%	83.80%	1.13E-03	9.91E-04	83.90%	83.70%
CD	83.81%	83.86%	83.77%	83.85%	83.93%	83.84%	6.01E-04	5.27E-04	83.90%	83.79%
RW	86.05%	86.32%	85.98%	86.09%	85.97%	86.08%	1.40E-03	1.23E-03	86.20%	85.96%
BT	82.39%	82.07%	82.19%	84.35%	82.22%	82.64%	9.58E-03	8.39E-03	83.48%	81.81%
Paraphrase	84.42%	84.35%	84.32%	84.35%	84.39%	84.36%	3.90E-04	3.42E-04	84.40%	84.33%
Average	84.96%	84.80%	84.81%	85.03%	84.87%	84.89%	1.01E-03	8.83E-04	84.98%	84.81%
Typos	83.79%	83.67%	83.71%	83.69%	83.76%	83.72%	8.36E-04	7.33E-04	83.80%	83.65%

Table B.35: Baseline retrieval accuracy at recall set size 200 Performance on TrivaQA with five random seeds with and without noisy queries.

Noise	Seed 1	Seed 2	Seed 3	Seed 3	Seed 5	Median	STD	CI	Upper Bound	Lower Bound
None	32.43%	32.41%	32.31%	32.81%	32.01%	32.39%	2.87E-03	2.51E-03	3.26E-01	3.21E-01
Determiner	25.86%	26.01%	25.55%	26.12%	25.71%	25.85%	2.30E-03	2.02E-03	2.61E-01	2.56E-01
Synonym	20.75%	20.52%	20.36%	20.94%	20.58%	20.63%	2.23E-03	1.96E-03	2.08E-01	2.04E-01
Lemma	31.55%	31.54%	31.50%	31.84%	31.10%	31.50%	2.65E-03	2.32E-03	3.17E-01	3.13E-01
Stem	26.88%	27.01%	26.65%	27.23%	26.59%	26.87%	2.63E-03	2.31E-03	2.71E-01	2.66E-01
RCS	16.87%	16.70%	16.46%	17.36%	16.89%	16.85%	3.29E-03	2.88E-03	1.71E-01	1.66E-01
KCS	15.93%	16.06%	15.63%	16.48%	15.88%	16.00%	3.14E-03	2.75E-03	1.63E-01	1.57E-01
CD	18.17%	17.91%	17.63%	18.46%	17.87%	18.00%	3.16E-03	2.77E-03	1.83E-01	1.77E-01
RW	31.37%	31.30%	30.99%	31.36%	30.96%	31.19%	2.05E-03	1.80E-03	3.14E-01	3.10E-01
BT	26.95%	26.57%	26.62%	27.10%	26.54%	26.76%	2.55E-03	2.23E-03	2.70E-01	2.65E-01
Paraphrase	26.69%	26.51%	26.38%	26.75%	26.50%	26.56%	1.55E-03	1.35E-03	2.67E-01	2.64E-01
Average	24.86%	24.78%	24.55%	25.13%	24.60%	24.78%	2.32E-03	2.03E-03	2.50E-01	2.46E-01
Typos	16.99%	16.89%	16.57%	17.43%	16.88%	16.95%	3.20E-03	2.80E-03	1.72E-01	1.67E-01

Table B.36: Dense Model MRR@10 Performance on MSMARCO with five random seeds with and without noisy queries.

Noise	Seed 1	Seed 2	Seed 3	Seed 3	Seed 5	Median	STD	CI	Upper Bound	Lower Bound
None	71.36%	72.03%	71.65%	71.83%	71.29%	71.63%	3.14E-03	2.75E-03	71.91%	71.36%
Determiner	59.64%	60.20%	59.91%	60.10%	59.46%	59.86%	3.11E-03	2.73E-03	60.14%	59.59%
Synonym	49.96%	50.09%	49.63%	50.46%	50.37%	50.10%	3.34E-03	2.93E-03	50.39%	49.81%
Lemma	69.84%	70.24%	92.71%	70.09%	69.99%	74.57%	1.01E-01	8.89E-02	83.46%	65.69%
Stem	61.91%	61.88%	61.70%	61.81%	79.23%	65.30%	7.78E-02	6.82E-02	72.13%	58.48%
RCS	41.33%	41.63%	40.87%	41.86%	40.80%	41.30%	4.63E-03	4.06E-03	41.71%	40.89%
KCS	39.66%	40.01%	39.18%	40.04%	39.20%	39.62%	4.20E-03	3.68E-03	39.99%	39.25%
CD	43.81%	44.77%	44.03%	44.80%	44.57%	44.40%	4.51E-03	3.95E-03	44.79%	44.00%
RW	69.84%	70.06%	69.63%	69.96%	69.91%	61.23%	2.03E-03	1.78E-03	61.40%	61.05%
BT	61.23%	61.40%	60.99%	61.45%	61.06%	65.05%	8.19E-02	7.18E-02	72.23%	57.87%
Paraphrase	61.28%	61.83%	79.70%	61.30%	61.15%	65.05%	8.19E-02	7.18E-02	72.23%	57.87%
Average	57.26%	57.65%	60.91%	57.61%	58.82%	58.45%	1.50E-02	1.31E-02	59.76%	57.14%
Typos	41.60%	42.14%	41.36%	42.23%	41.52%	41.77%	0.44%	0.39%	42.16%	41.38%

Table B.37: Baseline retrieval accuracy at recall set size 20 Performance on MSMARCO with five random seeds with and without noisy queries.

Noise	Seed 1	Seed 2	Seed 3	Seed 3	Seed 5	Median	STD	CI	Upper Bound	Lower Bound
None	88.40%	89.10%	88.67%	89.05%	88.75%	88.79%	2.90E-03	2.54E-03	89.05%	88.54%
Determiner	77.36%	77.69%	77.55%	77.89%	77.79%	77.66%	2.08E-03	1.83E-03	77.84%	77.48%
Synonym	67.88%	68.67%	67.52%	68.24%	68.14%	68.09%	4.26E-03	3.73E-03	68.46%	67.72%
Lemma	87.08%	87.81%	87.39%	87.85%	87.55%	87.54%	3.18E-03	2.79E-03	87.81%	87.26%
Stem	78.91%	79.76%	79.11%	79.50%	79.23%	79.30%	3.32E-03	2.91E-03	79.59%	79.01%
RCS	57.46%	58.38%	57.32%	58.90%	57.92%	58.00%	6.53E-03	5.72E-03	58.57%	57.43%
KCS	55.72%	56.07%	55.70%	56.79%	56.15%	56.09%	5.73E-03	5.02E-03	56.59%	55.58%
CD	61.49%	61.92%	61.78%	62.95%	61.69%	61.97%	3.56E-03	3.12E-03	62.28%	61.65%
RW	87.31%	87.84%	87.36%	87.77%	87.58%	87.57%	2.57E-03	2.25E-03	87.80%	87.35%
BT	78.35%	78.91%	78.12%	78.94%	78.68%	78.60%	3.56E-03	3.12E-03	78.91%	78.29%
Paraphrase	79.44%	79.74%	79.70%	80.04%	79.41%	79.67%	2.57E-03	2.25E-03	79.89%	79.44%
Average	74.49%	75.08%	74.57%	75.27%	74.81%	74.84%	3.31E-03	2.90E-03	75.13%	74.55%
Typos	58.22%	58.79%	58.27%	59.55%	58.59%	58.68%	5.27E-03	4.62E-03	59.14%	58.22%

Table B.38: Baseline retrieval accuracy at recall set size 100 Performance on MSMARCO with five random seeds with and without noisy queries.

Noise	Seed 1	Seed 2	Seed 3	Seed 3	Seed 5	Median	STD	CI	Upper Bound	Lower Bound
None	93.93%	93.95%	93.34%	93.90%	93.80%	93.78%	2.55E-03	2.24E-03	94.01%	93.56%
Determiner	83.38%	83.90%	83.44%	83.67%	83.44%	83.56%	2.16E-03	1.89E-03	83.75%	83.38%
Synonym	74.27%	75.03%	74.01%	74.76%	74.63%	74.54%	4.02E-03	3.52E-03	74.89%	74.19%
Lemma	92.68%	92.71%	92.35%	92.65%	92.48%	92.57%	1.54E-03	1.35E-03	92.71%	92.44%
Stem	84.74%	84.83%	84.30%	84.91%	84.70%	84.70%	2.37E-03	2.08E-03	84.90%	84.49%
RCS	63.95%	64.66%	63.81%	65.32%	64.36%	64.42%	6.02E-03	5.28E-03	64.95%	63.89%
KCS	62.05%	62.97%	62.61%	63.84%	62.91%	62.87%	6.51E-03	5.70E-03	63.44%	62.30%
CD	68.38%	68.47%	67.66%	68.88%	68.38%	68.36%	4.38E-03	3.84E-03	68.74%	67.97%
RW	92.58%	92.78%	92.56%	92.71%	92.52%	92.63%	1.08E-03	9.51E-04	92.73%	92.54%
BT	84.21%	84.36%	83.72%	83.97%	84.14%	84.08%	2.43E-03	2.13E-03	84.29%	83.87%
Paraphrase	85.60%	85.79%	85.70%	86.39%	85.60%	85.82%	3.30E-03	2.89E-03	86.11%	85.53%
Average	80.52%	80.86%	80.32%	81.00%	80.63%	80.67%	2.69E-03	2.36E-03	80.90%	80.43%
Typos	64.79%	65.36%	64.69%	66.01%	65.21%	65.22%	5.64E-03	4.94E-03	65.71%	64.72%

Table B.39: Baseline retrieval accuracy at recall set size 200 Performance on MSMARCO with five random seeds with and without noisy queries.

Noise	Regular	Loss	PT	Loss	DA	Loss	CAPOT	Loss
None	79.73%	-0.04%	75.04%	-5.88%	79.61%	-0.15%	77.84%	-2.37%
Determiner	74.59%	-6.49%	72.33%	-9.29%	77.67%	-2.58%	76.23%	-4.39%
Synonym	68.47%	-14.16%	67.12%	-15.81%	73.07%	-8.35%	71.66%	-10.12%
Lemma	74.33%	-6.82%	74.79%	-6.19%	77.95%	-2.23%	77.70%	-2.55%
Stem	74.24%	-6.92%	71.36%	-10.50%	77.95%	-2.23%	76.84%	-3.62%
RCS	67.12%	-15.84%	66.81%	-16.20%	75.24%	-5.64%	75.43%	-5.39%
KCS	67.01%	-15.98%	67.26%	-15.64%	75.82%	-4.91%	75.60%	-5.19%
CD	69.35%	-13.06%	67.48%	-15.37%	75.76%	-4.98%	75.54%	-5.26%
RW	76.46%	-4.14%	73.68%	-7.58%	78.39%	-1.68%	77.98%	-2.20%
BT	72.03%	-9.70%	67.23%	-15.68%	72.91%	-8.56%	71.27%	-10.61%
Paraphrase	72.00%	-9.73%	66.45%	-16.65%	73.05%	-8.38%	71.63%	-10.15%
Average	71.56%	-10.28%	69.45%	-12.89%	75.78%	-4.95%	74.99%	-5.95%
Typos	67.83%	-14.96%	67.18%	-15.74%	75.60%	-5.17%	75.52%	-5.28%

Table B.40: Retrieval accuracy and relative loss across types of noise for unaltered (Regular), PreTrained Alignment (PT), Data Augmentation (DA), and Post Training Contrastive Alignment (CAPOT) on NQ dataset with the recall set the size of 20

Noise	Regular	Loss	PT	Loss	DA	Loss	CAPOT	Loss
None	85.82%	-0.19%	84.60%	-1.61%	86.29%	0.36%	85.04%	-1.09%
Determiner	83.43%	-2.96%	81.91%	-4.73%	84.79%	-1.38%	83.66%	-2.70%
Synonym	79.75%	-7.25%	73.12%	-14.95%	81.94%	-4.70%	81.08%	-5.70%
Lemma	82.96%	-3.51%	84.27%	-1.99%	85.32%	-0.77%	84.96%	-1.19%
Stem	82.96%	-3.51%	82.08%	-4.54%	85.32%	-0.77%	84.68%	-1.51%
RCS	78.78%	-8.37%	78.95%	-8.18%	83.93%	-2.38%	83.49%	-2.90%
KCS	78.03%	-9.24%	79.11%	-7.99%	83.55%	-2.83%	83.32%	-3.09%
CD	79.64%	-7.37%	79.28%	-7.79%	84.29%	-1.96%	83.74%	-2.61%
RW	85.04%	-1.09%	83.74%	-2.61%	85.51%	-0.54%	85.07%	-1.06%
BT	80.11%	-6.83%	77.98%	-9.31%	80.53%	-6.34%	79.31%	-7.76%
Paraphrase	81.30%	-5.44%	78.78%	-8.37%	81.69%	-4.99%	80.78%	-6.05%
Average	81.62%	-5.07%	79.92%	-7.05%	83.69%	-2.67%	83.01%	-3.46%
Typos	78.82%	-8.33%	79.11%	-7.99%	83.92%	-2.39%	83.52%	-2.86%

Table B.41: Retrieval accuracy and relative loss across types of noise for unaltered (Regular), PreTrained Alignment (PT), Data Augmentation (DA), and Post Training Contrastive Alignment (CAPOT) on NQ dataset with the recall set the size of 100

Noise	Regular	Loss	PT	Loss	DA	Loss	CAPOT	Loss
None	88.25%	0.00%	87.01%	-1.41%	86.29%	-2.22%	87.23%	-1.16%
Determiner	85.99%	-2.56%	84.88%	-3.82%	86.73%	-1.72%	86.43%	-2.07%
Synonym	82.83%	-6.15%	82.11%	-6.96%	84.65%	-4.08%	84.07%	-4.73%
Lemma	85.13%	-3.54%	86.48%	-2.00%	87.37%	-1.00%	87.23%	-1.16%
Stem	85.72%	-2.87%	84.88%	-3.82%	87.37%	-1.00%	87.04%	-1.38%
RCS	81.96%	-7.13%	82.44%	-6.59%	85.96%	-2.60%	85.71%	-2.88%
KCS	81.52%	-7.63%	82.52%	-6.49%	85.82%	-2.76%	86.12%	-2.41%
CD	82.04%	-7.04%	82.41%	-6.62%	86.23%	-2.29%	86.65%	-1.82%
RW	87.34%	-1.03%	86.37%	-2.13%	87.40%	-0.97%	87.51%	-0.84%
BT	82.54%	-6.47%	80.64%	-8.63%	82.71%	-6.27%	82.05%	-7.03%
Paraphrase	84.09%	-4.72%	82.19%	-6.87%	83.91%	-4.92%	83.77%	-5.08%
Average	83.91%	-4.91%	83.49%	-5.39%	85.81%	-2.76%	85.66%	-2.94%
Typos	81.84%	-7.27%	82.46%	-6.57%	86.00%	-2.55%	86.16%	-2.37%

Table B.42: Retrieval accuracy and relative loss across types of noise for unaltered (Regular), PreTrained Alignment (PT), Data Augmentation (DA), and Post Training Contrastive Alignment (CAPOT) on NQ dataset with the recall set the size of 200

Noise	Regular	Loss	PT	Loss	DA	Loss	CAPOT	Loss
None	79.40%	0.00%	73.52%	-7.41%	76.16%	-4.08%	78.53%	-1.10%
Determiner	77.36%	-2.57%	71.54%	-9.90%	74.37%	-6.34%	77.76%	-2.07%
Synonym	75.11%	-5.41%	69.17%	-12.89%	73.22%	-7.79%	75.97%	-4.33%
Lemma	79.06%	-0.43%	73.41%	-7.54%	76.14%	-4.10%	78.69%	-0.90%
Stem	78.21%	-1.50%	72.57%	-8.60%	75.74%	-4.60%	78.24%	-1.46%
RCS	72.64%	-8.51%	67.60%	-14.86%	73.12%	-7.91%	76.74%	-3.35%
KCS	72.34%	-8.89%	67.87%	-14.52%	73.40%	-7.55%	76.90%	-3.15%
CD	73.14%	-7.88%	67.86%	-14.53%	73.31%	-7.68%	76.47%	-3.69%
RW	78.05%	-1.70%	72.80%	-8.31%	75.23%	-5.25%	78.74%	-0.83%
BT	74.05%	-6.74%	68.13%	-14.19%	70.65%	-11.02%	73.56%	-7.35%
Paraphrase	75.10%	-5.41%	68.66%	-13.52%	71.65%	-9.76%	74.19%	-6.56%
Average	75.51%	-4.90%	69.96%	-11.89%	73.68%	-7.20%	76.73%	-3.37%
Typos	72.71%	-8.43%	67.78%	-14.64%	73.28%	-7.71%	76.71%	-3.39%

Table B.43: Retrieval accuracy and relative loss across types of noise for unaltered (Regular), PreTrained Alignment (PT), Data Augmentation (DA), and Post Training Contrastive Alignment (CAPOT) on TriviaQA dataset with the recall set the size of 20

Noise	Regular	Loss	PT	Loss	DA	Loss	CAPOT	Loss
None	84.88%	-0.15%	81.70%	-3.89%	82.93%	-2.45%	84.85%	-0.19%
Determiner	83.70%	-1.54%	80.49%	-5.32%	81.71%	-3.88%	84.25%	-0.90%
Synonym	82.27%	-3.23%	78.51%	-7.64%	80.78%	-4.97%	83.12%	-2.23%
Lemma	84.68%	-0.39%	81.67%	-3.93%	82.85%	-2.54%	84.93%	-0.10%
Stem	84.20%	-0.95%	81.09%	-4.61%	82.67%	-2.75%	84.82%	-0.22%
RCS	80.92%	-4.81%	77.83%	-8.45%	81.23%	-4.44%	83.72%	-1.52%
KCS	81.19%	-4.49%	77.88%	-8.39%	81.29%	-4.38%	83.95%	-1.25%
CD	81.30%	-4.37%	78.22%	-7.99%	81.19%	-4.49%	83.75%	-1.48%
RW	84.20%	-0.95%	81.17%	-4.51%	82.56%	-2.88%	84.86%	-0.18%
BT	80.12%	-5.75%	76.72%	-9.76%	78.24%	-7.97%	80.16%	-5.70%
Paraphrase	82.21%	-3.30%	78.43%	-7.74%	79.80%	-6.13%	82.29%	-3.20%
Average	82.48%	-2.98%	79.20%	-6.83%	81.23%	-4.44%	83.58%	-1.68%
Typos	81.14%	-4.56%	77.98%	-8.28%	81.24%	-4.44%	83.81%	-1.42%

Table B.44: Retrieval accuracy and relative loss across types of noise for unaltered (Regular), PreTrained Alignment (PT), Data Augmentation (DA), and Post Training Contrastive Alignment (CAPOT) on TriviaQA dataset with the recall set the size of 100

Noise	Regular	Loss	PT	Loss	DA	Loss	CAPOT	Loss
None	86.66%	0.00%	84.03%	-3.04%	84.96%	-1.97%	86.60%	-0.07%
Determiner	85.68%	-1.13%	83.14%	-4.06%	84.13%	-2.92%	86.43%	-0.26%
Synonym	84.55%	-2.44%	81.43%	-6.04%	83.18%	-4.02%	85.32%	-1.55%
Lemma	86.50%	-0.18%	83.97%	-3.10%	84.89%	-2.04%	86.67%	0.01%
Stem	86.18%	-0.56%	83.59%	-3.55%	84.77%	-2.18%	86.56%	-0.11%
RCS	83.53%	-3.61%	81.13%	-6.38%	83.52%	-3.62%	85.91%	-0.87%
KCS	83.80%	-3.30%	81.18%	-6.32%	83.70%	-3.42%	85.99%	-0.77%
CD	83.84%	-3.25%	81.30%	-6.19%	83.60%	-3.53%	85.82%	-0.97%
RW	86.08%	-0.67%	83.74%	-3.37%	84.57%	-2.42%	86.69%	0.03%
BT	82.64%	-4.63%	79.41%	-8.36%	80.60%	-7.00%	82.37%	-4.95%
Paraphrase	84.36%	-2.65%	81.45%	-6.01%	82.66%	-4.62%	84.66%	-2.30%
Average	84.72%	-2.24%	82.03%	-5.34%	83.56%	-3.57%	85.64%	-1.17%
Typos	83.72%	-3.39%	81.20%	-5.47%	83.61%	-3.64%	85.91%	-0.87%

Table B.45: Retrieval accuracy and relative loss across types of noise for unaltered (Regular), PreTrained Alignment (PT), Data Augmentation (DA), and Post Training Contrastive Alignment (CAPOT) on TriviaQA dataset with the recall set the size of 200

Noise	Regular	Loss	PT	Loss	DA	Loss	CAPOT	Loss
None	32.39%	0.00%	19.05%	-41.18%	20.70%	-35.89%	25.38%	-21.39%
Determiner	25.85%	-20.20%	15.50%	-52.15%	16.42%	-49.15%	25.25%	-21.80%
Synonym	20.63%	-36.30%	11.56%	-64.30%	13.56%	-58.01%	17.92%	-44.49%
Lemma	31.50%	-2.74%	18.76%	-42.07%	20.66%	-36.03%	25.90%	-19.80%
Stem	26.87%	-17.04%	16.72%	-48.39%	18.68%	-42.14%	25.13%	-22.16%
RCS	16.85%	-47.96%	12.09%	-62.66%	15.13%	-53.16%	22.91%	-29.05%
KCS	16.00%	-50.62%	12.16%	-62.46%	14.86%	-53.97%	22.63%	-29.91%
CD	18.00%	-44.41%	12.52%	-61.36%	15.02%	-53.49%	22.46%	-30.45%
RW	31.19%	-3.70%	18.41%	-43.17%	19.91%	-38.35%	28.18%	-12.72%
BT	26.76%	-17.40%	15.52%	-52.08%	16.91%	-47.63%	20.80%	-35.58%
Paraphrase	26.56%	-17.99%	15.51%	-52.13%	16.82%	-47.90%	21.05%	-34.79%
Average	24.02%	-25.84%	14.87%	-54.08%	16.80%	-47.98%	23.22%	-28.08%
Typos	16.95%	-47.66%	12.26%	-62.16%	15.00%	-53.54%	22.67%	-29.80%

Table B.46: MRR@10 and relative loss across types of noise for unaltered (Regular), PreTrained Alignment (PT), Data Augmentation (DA), and Post Training Contrastive Alignment (CAPOT) on MSMARCO dataset.

Noise	Regular	Loss	PT	Loss	DA	Loss	CAPOT	Loss
None	32.39%	0.00%	19.05%	-41.18%	20.70%	-35.89%	25.38%	-21.39%
Determiner	25.85%	-20.20%	15.50%	-52.15%	16.42%	-49.15%	25.25%	-21.80%
Synonym	20.63%	-36.30%	11.56%	-64.30%	13.56%	-58.01%	17.92%	-44.49%
Lemma	31.50%	-2.74%	18.76%	-42.07%	20.66%	-36.03%	25.90%	-19.80%
Stem	26.87%	-17.04%	16.72%	-48.39%	18.68%	-42.14%	25.13%	-22.16%
RCS	16.85%	-47.96%	12.09%	-62.66%	15.13%	-53.16%	22.91%	-29.05%
KCS	16.00%	-50.62%	12.16%	-62.46%	14.86%	-53.97%	22.63%	-29.91%
CD	18.00%	-44.41%	12.52%	-61.36%	15.02%	-53.49%	22.46%	-30.45%
RW	31.19%	-3.70%	18.41%	-43.17%	19.91%	-38.35%	28.18%	-12.72%
BT	26.76%	-17.40%	15.52%	-52.08%	16.91%	-47.63%	20.80%	-35.58%
Paraphrase	26.56%	-17.99%	15.51%	-52.13%	16.82%	-47.90%	21.05%	-34.79%
Average	24.02%	-25.84%	14.87%	-54.08%	16.80%	-47.98%	23.22%	-28.08%
Typos	16.95%	-47.66%	12.26%	-62.16%	15.00%	-53.54%	22.67%	-29.80%

Table B.47: Retrieval accuracy and relative loss across types of noise for unaltered (Regular), PreTrained Alignment (PT), Data Augmentation (DA), and Post Training Contrastive Alignment (CAPOT) on MSMARCO dataset with the recall set the size of 20

Noise	Regular	Loss	PT	Loss	DA	Loss	CAPOT	Loss
None	88.79%	0.00%	67.87%	-23.57%	71.83%	-19.10%	79.70%	-10.24%
Determiner	77.66%	-12.54%	58.57%	-34.04%	63.02%	-29.02%	76.99%	-13.29%
Synonym	68.09%	-23.31%	47.12%	-46.93%	55.01%	-38.04%	61.96%	-30.21%
Lemma	87.54%	-1.41%	67.45%	-24.03%	71.73%	-19.21%	80.87%	-8.92%
Stem	79.30%	-10.69%	62.36%	-29.76%	68.05%	-23.36%	79.23%	-10.77%
RCS	58.00%	-34.68%	47.56%	-46.43%	58.81%	-33.76%	72.36%	-18.50%
KCS	56.09%	-36.83%	47.69%	-46.29%	59.03%	-33.52%	72.99%	-17.79%
CD	61.97%	-30.21%	49.99%	-43.70%	58.77%	-33.81%	71.99%	-18.92%
RW	87.57%	-1.37%	66.89%	-24.66%	70.74%	-20.32%	83.48%	-5.98%
BT	78.60%	-11.47%	57.97%	-34.72%	62.49%	-29.62%	68.91%	-22.39%
Paraphrase	79.67%	-10.27%	59.04%	-33.51%	63.74%	-28.21%	70.57%	-20.52%
Average	74.84%	-15.71%	56.46%	-36.41%	63.14%	-28.89%	73.94%	-16.73%
Typos	58.68%	-33.91%	48.41%	-45.47%	58.87%	-33.70%	72.45%	-18.40%

Table B.48: Retrieval accuracy and relative loss across types of noise for unaltered (Regular), PreTrained Alignment (PT), Data Augmentation (DA), and Post Training Contrastive Alignment (CAPOT) on MSMARCO dataset with the recall set the size of 100

Noise	Regular	Loss	PT	Loss	DA	Loss	CAPOT	Loss
None	93.78%	0.00%	74.99%	-20.04%	79.57%	-15.15%	85.69%	-8.63%
Determiner	83.56%	-10.89%	65.60%	-30.05%	70.56%	-24.76%	82.68%	-11.84%
Synonym	74.54%	-20.52%	54.54%	-41.84%	62.58%	-33.27%	68.72%	-26.72%
Lemma	92.57%	-1.29%	74.61%	-20.44%	79.31%	-15.43%	86.68%	-7.57%
Stem	84.70%	-9.69%	69.81%	-25.56%	75.62%	-19.37%	85.24%	-9.10%
RCS	64.42%	-31.31%	54.66%	-41.72%	66.40%	-29.19%	78.87%	-15.90%
KCS	62.87%	-32.96%	54.67%	-41.70%	67.32%	-28.21%	79.48%	-15.24%
CD	68.36%	-27.11%	56.81%	-39.43%	66.91%	-28.66%	78.11%	-16.71%
RW	92.63%	-1.23%	74.27%	-20.80%	78.04%	-16.79%	89.43%	-4.64%
BT	84.08%	-10.34%	65.56%	-30.09%	70.44%	-24.88%	75.27%	-19.74%
Paraphrase	85.82%	-8.49%	67.15%	-28.40%	71.35%	-23.92%	77.48%	-17.38%
Average	80.67%	-13.98%	63.77%	-32.00%	70.85%	-24.45%	80.20%	-14.48%
Typos	65.22%	-30.46%	55.38%	-40.95%	66.88%	-28.69%	78.82%	-15.95%

Table B.49: Retrieval accuracy and relative loss across types of noise for unaltered (Regular), PreTrained Alignment (PT), Data Augmentation (DA), and Post Training Contrastive Alignment (CAPOT) on MSMARCO dataset with the recall set the size of 200

Noise	Baseline	Loss	CAPOT	Loss	CAPOT-ORCAS	Loss
None	79.40%	0.00%	78.53%	-1.10%	76.73%	-3.36%
Determiner	77.36%	-2.57%	77.76%	-2.07%	76.04%	-4.24%
Synonym	75.11%	-5.41%	75.97%	-4.33%	81.23%	2.31%
Lemma	79.06%	-0.43%	78.69%	-0.90%	76.95%	-3.09%
Stem	78.21%	-1.50%	78.24%	-1.46%	76.68%	-3.42%
RCS	72.64%	-8.51%	76.74%	-3.35%	75.83%	-4.49%
KCS	72.34%	-8.89%	76.90%	-3.15%	75.62%	-4.76%
CD	73.14%	-7.88%	76.47%	-3.69%	75.28%	-5.19%
RW	78.05%	-1.70%	78.74%	-0.83%	76.81%	-3.27%
BT	74.05%	-6.74%	73.56%	-7.35%	71.48%	-9.97%
Paraphrase	75.10%	-5.41%	74.19%	-6.56%	71.95%	-9.38%
Average	75.51%	-4.90%	76.73%	-3.37%	75.79%	-4.55%
Typos	72.71%	-8.43%	76.71%	-3.39%	75.58%	-4.82%

Table B.50: Retrieval accuracy and relative loss across types of noise for unaltered (Regular), Post Training Contrastive Alignment (CAPOT), and Post Training Contrastive Alignment (CAPOT) using Noisy-ORCAS on TriviaQA dataset with the recall set the size of 20

Noise	Baseline	Loss	CAPOT	Loss	CAPOT-ORCAS	Loss
None	85.01%	0.00%	84.85%	-0.19%	85.63%	0.73%
Determiner	83.83%	-1.38%	84.25%	-0.90%	83.27%	-2.05%
Synonym	82.39%	-3.08%	83.12%	-2.23%	81.23%	-4.44%
Lemma	84.81%	-0.23%	84.93%	-0.10%	83.62%	-1.63%
Stem	84.36%	-0.77%	84.82%	-0.22%	83.58%	-1.69%
RCS	80.95%	-4.77%	83.72%	-1.52%	85.04%	0.04%
KCS	81.16%	-4.53%	83.95%	-1.25%	85.39%	0.45%
CD	81.33%	-4.33%	83.75%	-1.48%	82.66%	-2.77%
RW	82.99%	-2.37%	84.86%	-0.18%	83.56%	-1.71%
BT	80.60%	-5.19%	80.16%	-5.70%	78.77%	-7.34%
Paraphrase	82.40%	-3.07%	82.29%	-3.20%	80.33%	-5.50%
Average	82.48%	-2.97%	83.58%	-1.68%	82.74%	-2.66%
Typos	81.15%	-4.54%	83.81%	-1.42%	84.36%	-0.76%

Table B.51: Retrieval accuracy and relative loss across types of noise for unaltered (Regular), Post Training Contrastive Alignment (CAPOT), and Post Training Contrastive Alignment (CAPOT) using Noisy-ORCAS on TriviaQA dataset with the recall set the size of 100

Noise	Baseline	Loss	CAPOT	Loss	CAPOT-ORCAS	Loss
None	86.66%	0.00%	86.60%	-0.07%	83.52%	-3.62%
Determiner	85.68%	-1.13%	86.43%	-0.26%	85.49%	-1.36%
Synonym	84.55%	-2.44%	85.32%	-1.55%	83.68%	-3.44%
Lemma	86.50%	-0.18%	86.67%	0.01%	85.75%	-1.05%
Stem	86.18%	-0.56%	86.56%	-0.11%	85.57%	-1.25%
RCS	83.53%	-3.61%	85.91%	-0.87%	82.82%	-4.44%
KCS	83.80%	-3.30%	85.99%	-0.77%	83.09%	-4.12%
CD	83.84%	-3.25%	85.82%	-0.97%	84.88%	-2.05%
RW	86.08%	-0.67%	86.69%	0.03%	85.48%	-1.37%
BT	82.64%	-4.63%	82.37%	-4.95%	81.15%	-6.36%
Paraphrase	84.36%	-2.65%	84.66%	-2.30%	83.21%	-3.99%
Average	84.72%	-2.24%	85.64%	-1.17%	84.11%	-2.94%
Typos	83.72%	-3.39%	85.91%	-0.87%	83.60%	-3.53%

Table B.52: Retrieval accuracy and relative loss across types of noise for unaltered (Regular), Post Training Contrastive Alignment (CAPOT), and Post Training Contrastive Alignment (CAPOT) using Noisy-ORCAS on TriviaQA dataset with the recall set the size of 200

APPENDIX C: SCALING MULTI-LINGUAL CLASSIFICATION AND ABSTRACTIVE SUMMARIZATION TO WEB-SCALE WORKLOADS

C.1 TO ASYMMETRY AND BEYOND: STRUCTURED PRUNING OF SEQUENCE TO SEQUENCE MODELS FOR IMPROVED INFERENCE EFFICIENCY

C.1.1 Training Details

In all of our experiments, we leverage the parameters shown in C.2 on the datasets shown in C.1

Dataset	Train	Validation	Test	Source	Summary	Compression
CNNDM [76]	287,113	13,368	11,490	691.87	51.57	14.80
XSUM [77]	204,045	11,332	11,334	373.86	21.09	18.70
QIWS	10000	1000	1000	1410.12	73.78	19.11

Table C.1: Statistics for the abstractive summarization datasets which we study. Source and Summary refer to the number of words in each, and the compression factor is the ratio between the two on the train portion of the dataset.

C.1.2 Scale and Abstractive Summarization

The role of model scale on performance on the QIWS, CNN/DM, and XSUM datasets can be found in tables C.4,C.3, and C.5

	3,10 Epochs
Initial learning rate	1e-4
Learning rate schedule	constant
Batch size	64
Weight Decay	0.01, 0.05, 0.1

Table C.2: Training Hyperparameters for summarization experiments

Model	R-1	Impact	R-2	Impact	RSL	Impact	R-L	Impact	Genl	Impact
small	50.22	0.00%	29.03	0.00%	45.87	0.00%	40.19	0.00%	62.79	0.00%
base	54.84	9.20%	34.19	17.77%	50.38	9.83%	44.68	11.18%	62.91	0.19%
large	57.81	15.11%	37.37	28.72%	53.14	15.84%	48.16	19.84%	62.85	0.10%

Table C.3: Impact of Scale on summarization performance on QIWS dataset

Model	R-1	Impact	R-2	Impact	RSL	Impact	R-L	Impact	Genl	Impact
small	39.31	0.00%	17.55	0.00%	36.50	0.00%	27.97	0.00%	77.62	0.00%
base	42.14	7.20%	19.77	12.63%	39.32	7.75%	30.15	7.80%	71.86	-7.42%
large	43.99	11.90%	21.15	20.51%	41.12	12.68%	31.64	13.11%	71.01	-8.51%

Table C.4: Impact of Scale on summarization performance on CNNDM dataset

Model	R-1	Impact	R-2	Impact	RSL	Impact	R-L	Impact	Genl	Impact
small	33.27	0.00%	11.09	0.00%	26.17	0.00%	26.17	0.00%	28.01	0.00%
base	38.78	16.56%	15.69	41.45%	31.14	19.01%	31.15	19.04%	25.92	-7.48%
large	39.71	19.36%	16.34	47.36%	31.72	21.21%	31.72	21.23%	26.74	-4.54%

Table C.5: Impact of Scale on summarization performance on XSUM dataset

C.1.3 Asymmetry In Summarization

The role of the model scale, structural pruning, and asymmetry on performance on the QIWS, CNN/DM, and XSUM datasets can be found in tables C.12,C.13,C.14,C.6,C.7,C.8,C.9,C.10, and C.11.

l_{enc}	l_{dec}	R-1	Impact	R-2	Impact	RSL	Impact	R-L	Impact	GenL	Impact
8	8	39.31	0.00%	17.55	0.00%	36.50	0.00%	27.97	0.00%	77.62	0.00%
8	6	39.33	0.04%	17.68	0.74%	36.54	0.13%	28.21	0.87%	76.46	-1.49%
8	5	38.75	-1.42%	17.27	-1.64%	36.01	-1.32%	27.91	-0.23%	78.63	1.31%
8	4	37.18	-5.42%	16.40	-6.57%	34.46	-5.58%	27.22	-2.70%	75.69	-2.48%
8	2	35.47	-9.76%	15.35	-12.58%	32.78	-10.17%	26.28	-6.06%	75.08	-3.27%
8	1	29.27	-25.55%	11.33	-35.43%	26.97	-26.09%	22.33	-20.18%	67.99	-12.40%
6	8	39.59	0.71%	17.69	0.81%	36.80	0.83%	28.08	0.39%	77.81	0.25%
5	8	39.12	-0.47%	17.35	-1.16%	36.38	-0.31%	27.73	-0.88%	76.22	-1.80%
4	8	38.57	-1.87%	16.80	-4.30%	35.79	-1.92%	27.15	-2.92%	78.13	0.67%
2	8	36.82	-6.32%	15.54	-11.49%	34.00	-6.84%	25.79	-7.78%	77.77	0.20%
1	8	33.58	-14.58%	13.31	-24.17%	30.96	-15.16%	23.72	-15.19%	70.79	-8.79%
6	6	38.59	-1.82%	17.07	-2.77%	35.80	-1.91%	27.55	-1.52%	77.93	0.41%
5	5	37.31	-5.08%	16.20	-7.72%	34.60	-5.19%	26.83	-4.07%	79.83	2.85%
4	4	35.28	-10.25%	14.91	-15.05%	32.54	-10.85%	25.74	-7.98%	74.61	-3.88%
2	2	30.79	-21.66%	11.97	-31.83%	28.03	-23.19%	22.88	-18.19%	78.53	1.18%
1	1	21.30	-45.80%	6.05	-65.55%	19.57	-46.39%	16.62	-40.56%	60.03	-22.66%

Table C.6: The relation between pruning asymmetry and symmetry for a FLAN-T5 small model on the CNN/DailyMail Abstractive Summarization Dataset

C.1.4 Inference Benchmarks

Detailed variations in latency measurements across batch size, scale, structural pruning, and asymmetry on performance on the QIWS, CNN/DM, and XSUM datasets can be found

l_{enc}	l_{dec}	R-1	Impact	R-2	Impact	RSL	Impact	R-L	Impact	GenL	Impact
12	12	42.14	0.00%	19.77	0.00%	39.32	0.00%	30.15	0.00%	71.86	0.00%
12	10	42.49	0.84%	19.92	0.76%	39.62	0.75%	30.27	0.40%	74.38	3.51%
12	8	42.28	0.34%	19.85	0.42%	39.48	0.41%	30.35	0.64%	70.74	-1.56%
12	6	41.30	-1.99%	18.85	-4.63%	38.44	-2.25%	29.16	-3.28%	74.76	4.04%
12	4	40.31	-4.34%	18.68	-5.49%	37.71	-4.10%	29.45	-2.33%	67.52	-6.04%
12	2	36.75	-12.80%	16.48	-16.62%	34.22	-12.97%	27.61	-8.43%	67.67	-5.82%
10	12	42.49	0.84%	19.92	0.76%	39.62	0.75%	30.27	0.40%	74.38	3.51%
8	12	42.27	0.31%	19.67	-0.50%	39.41	0.22%	29.99	-0.52%	74.34	3.45%
6	12	41.30	-1.99%	18.85	-4.63%	38.44	-2.25%	29.16	-3.28%	74.76	4.04%
4	12	40.51	-3.86%	18.22	-7.86%	37.66	-4.23%	28.42	-5.75%	77.04	7.21%
2	12	39.03	-7.38%	17.06	-13.73%	36.15	-8.08%	27.23	-9.69%	73.36	2.09%
10	10	42.19	0.13%	19.72	-0.26%	39.38	0.14%	30.12	-0.11%	73.56	2.37%
8	8	41.64	-1.18%	19.17	-3.01%	38.83	-1.26%	29.60	-1.84%	74.59	3.80%
6	6	39.33	-6.67%	17.46	-11.71%	36.67	-6.74%	28.07	-6.92%	72.27	0.57%
4	4	36.99	-12.23%	15.87	-19.74%	34.43	-12.43%	26.63	-11.68%	69.08	-3.87%
2	2	30.99	-26.45%	12.23	-38.12%	28.43	-27.71%	23.28	-22.79%	66.70	-7.18%

Table C.7: The relation between pruning asymmetry and symmetry for a FLAN-T5 base model on the CNN/DailyMail Abstractive Summarization Dataset

l_{enc}	l_{dec}	R-1	Impact	R-2	Impact	RSL	Impact	R-L	Impact	GenL	Impact
24	24	43.99	0.00%	21.15	0.00%	41.12	0.00%	31.64	0.00%	71.01	0.00%
24	20	44.15	0.37%	21.30	0.69%	41.31	0.46%	31.73	0.31%	71.20	0.26%
24	16	44.10	0.27%	21.32	0.81%	41.29	0.39%	31.83	0.60%	70.19	-1.16%
24	12	43.74	-0.57%	21.08	-0.34%	40.97	-0.38%	31.60	-0.13%	69.99	-1.44%
24	8	43.35	-1.45%	20.67	-2.27%	40.58	-1.32%	31.29	-1.11%	72.88	2.63%
24	4	41.42	-5.84%	19.49	-7.88%	38.78	-5.69%	30.35	-4.06%	70.39	-0.89%
20	24	44.10	0.26%	21.13	-0.12%	41.28	0.38%	31.58	-0.17%	71.04	0.04%
16	24	43.76	-0.52%	20.83	-1.53%	40.92	-0.49%	31.22	-1.31%	71.59	0.80%
12	24	43.33	-1.50%	20.53	-2.94%	40.43	-1.68%	30.82	-2.58%	73.28	3.20%
8	24	42.46	-3.48%	19.74	-6.67%	39.64	-3.60%	29.98	-5.23%	73.47	3.46%
4	24	41.25	-6.23%	18.68	-11.69%	38.30	-6.86%	28.78	-9.04%	76.05	7.08%
20	20	44.10	0.25%	21.23	0.34%	41.25	0.32%	31.65	0.05%	70.90	-0.16%
16	16	43.69	-0.67%	20.90	-1.19%	40.86	-0.64%	31.30	-1.06%	71.85	1.18%
12	12	42.81	-2.67%	20.13	-4.84%	39.97	-2.80%	30.58	-3.33%	72.81	2.53%
8	8	40.57	-7.78%	18.47	-12.70%	37.82	-8.04%	28.96	-8.46%	73.39	3.34%
4	4	36.11	-17.91%	15.51	-26.68%	33.48	-18.59%	26.30	-16.88%	68.58	-3.43%

Table C.8: The relation between pruning asymmetry and symmetry for a FLAN-T5 large model on the CNN/DailyMail Abstractive Summarization Dataset

l_{enc}	l_{dec}	R-1	Impact	R-2	Impact	RSL	Impact	R-L	Impact	GenL	Impact
8	8	50.22	100.00%	29.03	100.00%	45.87	100.00%	40.19	100.00%	62.79	100.00%
8	6	50.20	99.96%	28.90	99.55%	45.80	99.83%	40.45	100.65%	62.81	100.03%
8	5	49.74	99.04%	28.56	98.40%	45.55	99.30%	40.27	100.20%	62.68	99.83%
8	4	48.59	96.74%	27.94	96.24%	44.65	97.33%	39.27	97.70%	62.67	99.82%
8	2	45.36	90.32%	24.85	85.61%	41.38	90.21%	36.92	91.87%	62.68	99.84%
8	1	34.47	68.64%	15.41	53.08%	31.00	67.58%	27.68	68.88%	61.68	98.24%
6	8	49.32	98.21%	27.92	96.17%	44.72	97.48%	39.10	97.28%	62.90	100.18%
5	8	49.08	97.72%	27.75	95.60%	44.29	96.56%	38.76	96.45%	62.87	100.13%
4	8	46.40	92.39%	25.20	86.82%	41.81	91.14%	36.71	91.34%	62.74	99.93%
2	8	45.08	89.77%	23.67	81.55%	40.44	35.31%	35.31	87.85%	62.82	100.06%
1	8	39.81	79.26%	18.23	62.79%	35.39	77.14%	29.97	74.56%	62.83	100.07%
6	6	48.47	96.51%	26.82	92.38%	43.88	95.66%	38.38	95.49%	62.81	100.04%
5	5	47.55	94.68%	26.62	91.72%	43.13	94.02%	37.99	94.51%	62.67	99.81%
4	4	42.33	84.28%	23.12	79.64%	39.89	86.95%	33.39	83.08%	62.71	99.88%
2	2	39.69	79.02%	19.14	65.92%	35.49	77.36%	30.90	76.89%	62.79	100.00%
1	1	22.98	45.75%	6.09	20.99%	20.52	44.74%	18.36	45.69%	61.90	98.58%

Table C.9: The relation between pruning asymmetry and symmetry for a FLAN-T5 small model on the Query Independent Web Snippets Abstractive Summarization Dataset

in tables C.15,C.16, C.17, C.18,C.19, C.20, C.23, C.21, and C.22.

C.1.5 Responsible NLP Research - Reproducibility Checklist

Datasets. We perform our experimentation on well-established benchmarks using many broad domains and a proprietary web summarization dataset. We do not modify or augment

l_{enc}	l_{dec}	R-1	Impact	R-2	Impact	RSL	Impact	R-L	Impact	GenL	Impact
12	12	54.84	0.00%	34.19	0.00%	50.38	0.00%	44.68	0.00%	62.91	0.00%
12	10	55.02	0.33%	34.00	-0.56%	50.20	-0.35%	44.67	-0.02%	62.79	-0.19%
12	8	55.97	2.05%	34.50	0.91%	51.12	1.48%	44.90	0.48%	62.75	-0.24%
12	6	54.54	-0.55%	33.70	-1.42%	49.94	-0.87%	44.19	-1.11%	62.81	-0.16%
12	4	52.64	-4.01%	31.93	-6.62%	47.28	-6.16%	42.98	-3.81%	62.85	-0.09%
12	2	49.02	-10.61%	28.05	-17.97%	44.98	-10.71%	40.36	-9.68%	62.89	-0.02%
10	12	54.23	-1.11%	33.57	-1.82%	49.93	-0.89%	44.00	-1.52%	62.87	-0.05%
8	12	54.02	-1.50%	33.06	-3.31%	49.49	-1.76%	43.80	-1.96%	62.85	-0.09%
6	12	48.74	-11.13%	32.23	-5.72%	48.74	-3.26%	42.92	-3.95%	62.82	-0.14%
4	12	47.93	-12.61%	27.47	-19.65%	46.21	-8.28%	39.77	-11.00%	62.79	-0.19%
2	12	47.45	-13.48%	25.57	-25.22%	43.20	-14.26%	37.69	-15.66%	62.77	-0.22%
10	10	54.25	-1.08%	32.88	-3.82%	49.51	-1.72%	43.24	-3.23%	62.82	-0.13%
8	8	53.89	-1.73%	32.81	-4.04%	49.32	-2.10%	43.77	-2.04%	62.82	-0.14%
6	6	50.26	-8.34%	28.70	-16.05%	45.62	-9.45%	40.05	-10.37%	62.82	-0.13%
4	4	47.77	-12.89%	26.53	-22.40%	43.34	-13.97%	37.85	-15.29%	62.84	-0.10%
2	2	39.59	-27.80%	19.64	-42.57%	35.80	-28.95%	31.38	-29.78%	62.85	-0.09%

Table C.10: The relation between pruning asymmetry and symmetry for a FLAN-T5 base model on the Query Independent Web Snippets Abstractive Summarization Dataset

l_{enc}	l_{dec}	R-1	Impact	R-2	Impact	RSL	Impact	R-L	Impact	GenL	Impact
24	24	57.81	100.00%	37.37	100.00%	53.14	100.00%	48.16	100.00%	62.85	100.00%
24	20	58.21	100.69%	37.59	100.59%	53.44	100.58%	48.46	100.62%	62.80	99.91%
24	16	57.25	99.04%	36.56	97.84%	52.71	99.19%	47.71	99.06%	62.83	99.97%
24	12	56.78	98.21%	35.74	95.64%	52.34	98.49%	46.81	97.18%	62.78	99.88%
24	8	56.19	97.19%	35.13	94.01%	51.59	97.08%	45.68	94.85%	62.79	99.90%
24	4	54.53	94.32%	33.69	90.15%	50.00	94.10%	44.65	92.71%	62.83	99.97%
20	24	57.34	99.19%	36.39	97.38%	52.66	99.10%	47.28	98.18%	62.81	99.93%
16	24	56.26	97.33%	35.90	96.07%	51.04	96.04%	46.82	97.22%	62.81	99.93%
12	24	55.31	95.67%	34.22	91.58%	50.60	95.23%	45.11	93.66%	62.88	100.04%
8	24	54.80	94.79%	33.42	89.43%	49.95	94.00%	44.11	91.59%	62.70	99.76%
4	24	51.40	88.92%	30.31	81.11%	46.49	87.48%	41.12	85.38%	62.70	99.75%
20	20	56.81	98.28%	36.32	97.20%	52.21	98.25%	46.82	97.21%	62.69	99.74%
16	16	56.10	97.05%	35.98	96.29%	51.05	96.07%	45.89	95.28%	62.71	99.76%
12	12	54.16	93.70%	33.00	88.31%	49.58	93.31%	44.80	93.02%	62.77	99.87%
8	8	51.77	89.55%	30.78	82.38%	47.31	89.03%	41.32	85.79%	62.73	99.81%
4	4	45.70	79.06%	22.77	60.94%	41.36	77.84%	36.09	74.94%	62.70	99.76%

Table C.11: The relation between pruning asymmetry and symmetry for a FLAN-T5 large model on the Query Independent Web Snippets Abstractive Summarization Dataset

l_{enc}	l_{dec}	R-1	Impact	R-2	Impact	RSL	Impact	R-L	Impact	GenL	Impact
8	8	33.27	0.00%	11.09	0.00%	26.17	0.00%	26.17	0.00%	28.01	0.00%
8	6	33.79	1.56%	11.61	4.74%	26.73	2.14%	26.74	2.18%	27.79	-0.78%
8	5	33.47	0.61%	11.43	3.12%	26.64	1.81%	26.65	1.83%	27.40	-2.18%
8	3	33.04	-0.69%	11.24	1.36%	26.26	0.36%	26.27	0.38%	28.08	0.26%
8	2	31.48	-5.36%	10.53	-5.02%	25.39	-2.99%	25.38	-3.01%	26.58	-5.13%
8	1	23.16	-30.39%	6.03	-45.58%	19.02	-27.32%	19.02	-27.33%	36.68	30.93%
5	8	33.31	0.13%	11.18	0.82%	26.16	-0.04%	26.16	-0.06%	28.31	1.08%
5	8	32.55	-2.15%	10.61	-4.32%	25.50	-2.55%	25.50	-2.55%	28.35	1.19%
3	8	31.82	-4.36%	10.11	-8.84%	24.92	-4.78%	24.92	-4.77%	28.43	1.50%
2	8	29.65	-10.87%	8.59	-22.48%	23.02	-12.02%	23.02	-12.03%	27.90	-0.39%
1	8	28.46	-14.46%	7.70	-30.57%	22.09	-15.60%	22.09	-15.59%	27.87	-0.50%
6	6	32.50	-2.29%	10.73	-3.24%	25.67	-1.90%	25.68	-1.88%	28.07	0.19%
5	5	31.77	-4.50%	10.19	-8.04%	25.14	-3.94%	25.14	-3.95%	28.09	0.29%
3	3	30.42	-8.57%	9.50	-14.31%	24.16	-7.66%	24.16	-7.67%	27.91	-0.38%
2	2	26.71	-19.70%	7.31	-34.09%	21.38	-18.30%	21.38	-18.31%	26.35	-5.93%
1	1	19.54	-41.26%	4.00	-63.91%	16.00	-38.86%	16.00	-38.87%	35.73	27.54%

Table C.12: The relation between pruning asymmetry and symmetry for a FLAN-T5 small model on the Extreme Summarization (XSUM) Abstractive Summarization Dataset

public benchmarks in any dataset.

Models. The model used as a starting point for all of our experiments is the family of flan-t5 models, publicly available via HuggingFace Hub [78]. All other models presented in this section are openly-available in the hugging face hub.

[78]https://huggingface.co/bert-base-uncased

l_{enc}	l_{dec}	R-1	Impact	R-2	Impact	RSL	Impact	R-L	Impact	GenL	Impact
12	12	38.78	0.00%	15.69	0.00%	31.14	0.00%	31.15	0.00%	25.92	0.00%
12	10	38.46	-0.83%	15.27	-2.65%	30.70	-1.43%	30.71	-1.42%	26.72	3.11%
12	8	38.11	-1.72%	14.91	-4.97%	30.34	-2.59%	30.34	-2.60%	27.64	6.65%
12	6	38.55	-0.58%	15.40	-1.83%	30.87	-0.87%	30.88	-0.87%	27.42	5.80%
12	4	38.04	-1.91%	15.19	-3.18%	30.63	-1.64%	29.65	-4.82%	26.40	1.85%
12	2	35.39	-8.74%	13.73	-12.47%	28.96	-7.02%	28.96	-7.03%	27.55	6.32%
10	12	39.04	0.68%	15.92	1.47%	31.22	0.24%	31.23	0.25%	26.89	3.75%
8	12	37.05	-4.45%	14.10	-10.09%	29.29	-5.95%	29.30	-5.93%	27.68	6.82%
6	12	36.45	-6.01%	13.84	-11.79%	28.96	-7.02%	28.96	-7.02%	27.21	4.99%
4	12	34.32	-11.48%	12.10	-22.88%	26.99	-13.35%	26.99	-13.34%	27.20	4.94%
2	12	31.88	-17.78%	10.27	-34.53%	24.85	-20.21%	24.85	-20.22%	28.22	8.88%
10	10	38.80	0.05%	15.72	0.22%	31.07	-0.25%	31.08	-0.23%	26.92	3.88%
8	8	37.21	-4.04%	14.30	-8.85%	29.55	-5.13%	29.54	-5.15%	27.40	5.72%
6	6	34.92	-9.95%	12.44	-20.68%	27.56	-11.51%	27.57	-11.50%	27.72	6.96%
4	4	32.48	-16.24%	10.67	-31.95%	25.49	-18.15%	25.50	-18.14%	27.98	7.98%
2	2	27.44	-29.23%	7.74	-50.65%	21.95	-29.51%	21.96	-29.52%	29.38	13.38%

Table C.13: The relation between pruning asymmetry and symmetry for a FLAN-T5 base model on the Extreme Summarization (XSUM) Abstractive Summarization Dataset

l_{enc}	l_{dec}	R-1	Impact	R-2	Impact	RSL	Impact	R-L	Impact	GenL	Impact
24	24	39.71	0.00%	16.34	0.00%	31.72	0.00%	31.72	0.01%	26.74	0.00%
24	20	43.18	8.74%	19.80	21.17%	35.21	11.01%	35.22	11.04%	25.91	-3.10%
24	16	42.73	7.59%	19.30	18.10%	34.76	9.58%	34.76	9.59%	26.40	-1.29%
24	12	42.34	6.61%	18.92	15.78%	34.52	8.84%	34.53	8.87%	25.49	-4.68%
24	8	41.30	4.00%	17.96	9.94%	33.73	6.34%	33.75	6.39%	25.02	-6.45%
24	4	39.55	-0.40%	16.47	0.77%	32.25	1.66%	32.25	1.68%	26.30	-1.64%
20	24	42.77	7.71%	19.43	18.90%	34.83	9.82%	34.84	9.83%	26.18	-2.09%
16	24	41.55	4.63%	18.33	12.17%	33.64	6.05%	33.65	6.07%	26.33	-1.53%
12	24	39.95	0.61%	16.90	3.40%	32.13	1.29%	32.14	1.31%	27.14	1.50%
8	24	37.57	-5.39%	14.97	-8.38%	29.94	-5.61%	29.94	-5.60%	25.99	-2.80%
4	24	34.81	-12.35%	12.52	-23.36%	27.32	-13.86%	27.32	-13.86%	27.61	3.25%
20	20	42.48	6.98%	19.18	17.39%	34.62	9.13%	34.62	9.13%	25.84	-3.36%
16	16	40.78	2.69%	17.56	7.44%	32.99	4.00%	33.00	4.02%	26.47	-1.00%
12	12	38.94	6.98%	15.89	-2.78%	31.21	-1.61%	31.22	-1.58%	26.59	-0.57%
8	8	34.65	-12.75%	12.15	-25.65%	27.36	-13.76%	27.36	-13.73%	28.16	5.30%
4	4	29.82	-24.91%	8.96	-45.14%	23.59	-25.62%	23.60	-25.60%	28.10	5.09%

Table C.14: The relation between pruning asymmetry and symmetry for a FLAN-T5 large model on the Extreme Summarization (XSUM) Abstractive Summarization Dataset

l_{enc}	l_{dec}	R-2	Impact	BS 1	STD	Speedup	BS 8	STD	Speedup	BS 16	STD	Speedup
8	8	29.03	0.00%	524	3.95	1.00	653	2.49	1.00	729	5.12	1.00
8	6	28.90	-0.45%	406	1.28	1.29	514	5.02	1.27	583	2.47	1.25
8	5	28.56	-1.60%	348	2.34	1.51	455	1.6	1.44	527	1.85	1.38
8	4	27.94	-3.76%	293	3.35	1.79	394	6.32	1.66	469	2.65	1.55
8	2	24.85	-14.39%	195	1.61	2.69	353	3.38	1.85	426	6.38	1.71
8	1	15.41	-46.92%	132	0.959	3.97	211	2.82	3.09	389	2.94	1.87
6	8	27.92	-3.83%	512	5.15	1.02	626	4.19	1.04	684	2.81	1.07
5	8	27.75	-4.40%	508	3.56	1.03	617	4.91	1.06	666	4.16	1.09
4	8	25.20	-13.18%	514	3.55	1.02	603	4.52	1.08	639	2.08	1.14
2	8	23.67	-18.45%	514	514	1.02	585	5.36	1.12	608	4.45	1.20
1	8	18.23	-37.21%	510	5.81	1.03	574	4.21	1.14	595	7.06	1.23
6	6	26.82	-7.62%	407	5.26	1.29	496	8.77	1.32	548	1.97	1.33
5	5	26.62	-8.28%	346	6.84	1.51	430	3.54	1.52	480	12.4	1.52
4	4	23.12	-20.36%	375	4.25	1.40	441	6.92	1.48	478	10.6	1.53
2	2	19.14	-34.08%	402	2.05	1.30	452	9.84	1.44	476	8.29	1.53
1	1	6.09	-79.01%	134	6.2	3.91	527	3.03	1.24	549	13.4	1.33

Table C.15: Role of model symmetry in inference efficiency on FLAN-T5 small model on the QIWS dataset

C.1.6 Computational Experiments

Our experimentation on finetuning our compressed models uses a single 40GB A100. Finetuning time varies across datasets ranging from 1 hour for T5-small to 24 hours for T5-Large.

l_{enc}	l_{dec}	R-2	Impact	BS 1	STD	Speedup	BS 8	STD	Speedup	BS 16	STD	Speedup
12	12	34.19	0.00%	746	11	1.00	1060	2.84	1.00	1310	6.8	1.00
12	10	34.00	-0.56%	625	3.27	1.19	943	4.69	1.12	1200	4.8	1.09
12	8	34.50	0.91%	523	2.19	1.43	814	4.23	1.30	1070	5.34	1.22
12	6	33.70	-1.42%	425	1.92	1.76	652	3.39	1.63	970	4.79	1.35
12	4	31.93	-6.62%	350	1.32	2.13	510	3.1	2.08	815	2	1.61
12	2	28.05	-17.97%	202	1.41	3.69	451	2.92	2.35	762	0.911	1.72
10	12	33.57	-1.82%	710	6.2	1.05	995	2.74	1.07	1290	4.2	1.02
8	12	33.06	-3.31%	690	5.72	1.08	953	5.72	1.11	1270	4.3	1.03
6	12	32.23	-5.72%	716	8	1.04	944	7.22	1.12	1080	5.29	1.21
4	12	27.47	-19.65%	710	1.75	1.05	911	10.1	1.16	1,000	8.84	1.31
2	12	25.57	-25.22%	706	5.4	1.06	862	7.11	1.23	921	7.04	1.42
10	10	32.88	-3.82%	633	11.6	1.18	915	11	1.16	1120	5.51	1.17
8	8	32.81	-4.04%	512	4.98	1.46	737	9.78	1.44	911	4.98	1.44
6	6	28.70	-16.05%	401	3.16	1.86	572	4.73	1.85	702	1.57	1.87
4	4	26.53	-22.40%	301	2.92	2.48	415	3.01	2.55	509	0.997	2.57
2	2	19.64	-42.57%	189	1.98	3.95	312	2.88	3.40	389	0.892	3.37

Table C.16: Role of model symmetry in inference efficiency on FLAN-T5 base model on the QIWS dataset

l_{enc}	l_{dec}	R-2	Impact	BS 1	STD	Speedup	BS 8	STD	Speedup	BS 16	STD	Speedup
24	24	37.37	0.00%	1430	6.08	1.00	2240	4.81	1.00	3320	1.02	1.00
24	20	37.59	0.59%	1210	4.73	1.18	1990	6.89	1.13	3010	2.63	1.10
24	16	36.56	-2.16%	1000	2.70	1.43	1750	5.92	1.28	2710	1.57	1.23
24	12	35.74	-4.36%	795	6.61	1.80	1510	10.40	1.48	2400	1.59	1.38
24	8	35.13	-5.99%	585	4.99	2.44	1260	7.14	1.78	2090	7.17	1.59
24	4	33.69	-9.85%	373	1.16	3.83	1030	10.50	2.17	1790	1.72	1.85
20	24	36.39	-2.62%	1410	3.66	1.01	2130	10.90	1.05	3090	5.98	1.07
16	24	35.90	-3.93%	1395	3.52	1.03	2060	9.89	1.09	2880	3.32	1.15
12	24	34.22	-8.42%	1380	5.20	1.04	1900	9.65	1.18	2630	0.81	1.26
8	24	33.42	-10.57%	1370	5.49	1.04	1790	19.00	1.25	2400	1.34	1.38
4	24	30.31	-18.89%	1350	7.33	1.06	1670	5.30	1.34	2170	2.79	1.53
20	20	36.32	-2.80%	1200	5.37	1.19	1880	7.89	1.19	2780	1.15	1.19
16	16	35.98	-3.71%	1020	3.49	1.40	1530	5.62	1.46	2230	1.80	1.49
12	12	33.00	-11.69%	749	5.30	1.91	1160	2.94	1.93	1710	0.89	1.94
8	8	30.78	-17.62%	650	3.32	2.20	970	2.78	2.31	1550	0.79	2.14
4	4	22.77	-39.06%	585	2.23	2.44	890	3.21	2.52	1450	0.92	2.29

Table C.17: Role of model symmetry in inference efficiency on FLAN-T5 large model on the QIWS dataset

l_{enc}	l_{dec}	R-2	Impact	BS 1	STD	Speedup	BS 8	STD	Speedup	BS 16	STD	Speedup
8	8	17.55	0.00%	138	5.05	1.00	230	7.61	1.00	330	3.71	1.00
8	6	17.68	0.74%	133	0.292	1.04	211	0.425	1.09	300	0.954	1.10
8	5	17.27	-1.64%	116	0.196	1.19	193	0.448	1.19	279	0.537	1.18
8	4	16.40	-6.57%	98.1	0.242	1.41	174	0.153	1.32	259	0.424	1.27
8	2	15.35	-12.58%	63.2	0.207	2.18	137	0.1	1.68	218	0.303	1.51
8	1	11.33	-35.43%	45.7	0.106	3.02	118	0.0827	1.95	198	0.148	1.67
6	8	17.69	0.81%	166	0.303	0.83	230	1.42	1.00	303	1.06	1.09
5	8	17.35	-1.16%	165	0.267	0.84	219	0.521	1.05	283	1.13	1.17
4	8	16.80	-4.30%	164	0.185	0.84	211	0.89	1.09	265	1.85	1.25
2	8	15.54	-11.49%	162	332	0.85	191	0.332	1.20	226	625	1.46
1	8	13.31	-24.17%	161	0.626	0.86	180	0.423	1.28	206	0.55	1.60
6	6	17.07	-2.77%	131	0.617	1.05	192	0.247	1.20	261	0.768	1.26
5	5	16.20	-7.72%	113	0.306	1.22	164	0.642	1.40	220	1.36	1.50
4	4	14.91	-15.05%	95.1	0.0955	1.45	135	0.21	1.70	182	0.268	1.81
2	2	11.97	-31.83%	57.8	0.27	2.39	78.9	0.078	2.92	103	0.238	3.20
1	1	6.05	-65.55%	39.1	0.136	3.53	50.2	0.132	4.58	63.4	0.0845	5.21

Table C.18: Role of model symmetry in inference efficiency on FLAN-T5 small model on the CNNDM dataset

C.1.7 Computational Packages

All of our experimentation is done using public libraries and datasets to ensure extensibility and reproducibility. Our investigation is done using HuggingFace's Transformers [79]

[79]https://github.com/huggingface/transformers

l_{enc}	l_{dec}	R-2	Impact	BS 1	STD	Speedup	BS 8	STD	Speedup	BS 16	STD	Speedup
12	12	19.77	0.00%	199	3.74	1.00	550	3.81	1.00	931	2.09	1.00
12	10	19.92	0.76%	179	3.31	1.11	524	16.2	1.05	889	4.41	1.05
12	8	19.85	0.42%	155	4.50	1.28	493	14	1.12	884	3.61	1.05
12	6	18.85	-4.63%	126	1.95	1.58	449	5.88	1.22	800	4.59	1.16
12	4	18.68	-5.49%	99.2	1.02	2.01	405	1.41	1.36	737	5.06	1.26
12	2	16.48	-16.62%	75.3	0.85	2.64	372	1.98	1.48	697	4.55	1.34
10	12	19.92	0.76%	198	4.75	1.01	495	14.5	1.11	811	1.18	1.15
8	12	19.67	-0.50%	196	3.72	1.02	441	7.82	1.25	715	4.39	1.30
6	12	18.85	-4.63%	187	4.81	1.06	396	13.3	1.39	613	9.45	1.52
4	12	18.22	-7.86%	183	3.54	1.09	330	5.04	1.67	509	2.1	1.83
2	12	17.06	-13.73%	176	3.52	1.13	272	1.79	2.02	400	3.25	2.33
10	10	19.72	-0.26%	171	3.21	1.16	462	11.9	1.19	776	4.62	1.20
8	8	19.17	-3.01%	141	2.97	1.41	37	12.1	14.86	628	6.48	1.48
6	6	17.46	-11.71%	109	1.71	1.83	281	2.61	1.96	478	3.55	1.95
4	4	15.87	-19.74%	82.5	1.24	2.41	198	1.71	2.78	329	0.74	2.83
2	2	12.23	-38.12%	50.7	1.30	3.93	112	2.59	4.91	178	0.557	5.23

Table C.19: Role of model symmetry in inference efficiency on FLAN-T5 base model on the CNNDM dataset

l_{enc}	l_{dec}	R-2	Impact	BS 1	STD	Speedup	BS 8	STD	Speedup	BS 16	STD	Speedup
24	24	21.15	0.00%	445	2.35	1.00	1480	20.1	1.00	2700	7.22	1.00
24	20	21.30	0.69%	390	33.7	1.14	1390	4.24	1.06	2590	7.7	1.04
24	16	21.32	0.81%	335	13.9	1.33	1330	7.7	1.11	2470	7.42	1.09
24	12	21.08	-0.34%	270	3.28	1.65	1250	11	1.18	2340	6.68	1.15
24	8	20.67	-2.27%	219	8.67	2.03	1180	8.17	1.25	2220	4.25	1.22
24	4	19.49	-7.88%	165	1.81	2.70	1090	6.6	1.36	2090	9.15	1.29
20	24	21.13	-0.12%	418	13.8	1.06	1320	15.3	1.12	2400	7.26	1.13
16	24	20.83	-1.53%	421	16.8	1.06	1150	16	1.29	2080	6.07	1.30
12	24	20.53	-2.94%	391	12.5	1.14	1000	21.7	1.48	1750	8.18	1.54
8	24	19.74	-6.67%	373	13.1	1.19	882	6.92	1.68	1430	4.79	1.89
4	24	18.68	-11.69%	350	4.32	1.27	670	15	2.21	1110	3.21	2.43
20	20	21.23	0.34%	359	4.3	1.24	1240	15.3	1.19	2260	6.73	1.19
16	16	20.90	-1.19%	1289	2.5	0.35	994	21.6	1.49	1820	4.27	1.48
12	12	20.13	-4.84%	229	12.1	1.94	756	12.6	1.96	1370	4.6	1.97
8	8	18.47	-12.70%	160	31.8	2.78	513	2.55	2.88	926	7.24	2.92
4	4	15.51	-26.68%	89.7	0.588	4.96	267	2.14	5.54	479	4.3	5.64

Table C.20: Role of model symmetry in inference efficiency on FLAN-T5 LARGE model on the CNNDM dataset

l_{enc}	l_{dec}	R-2	Impact	BS 1	STD	Speedup	BS 8	STD	Speedup	BS 16	STD	Speedup
8	8	11.09	0.00%	135	2.73	1.00	227	3.51	1.00	332	1.91	1.00
8	6	11.61	4.74%	108	1.70	1.25	196	1.94	1.16	303	7.95	1.10
8	5	11.43	3.12%	94.1	3.02	1.43	183	3.43	1.24	281	6.77	1.18
8	4	11.24	1.36%	82.7	2.66	1.63	168	2.33	1.35	263	2.24	1.26
8	2	10.53	-5.02%	55.8	1.72	2.42	141	1.53	1.61	234	5.01	1.42
8	1	6.03	-45.58%	41.1	0.64	3.28	124	0.414	1.83	215	4.69	1.54
6	8	11.18	0.82%	133	3.51	1.02	204	3.63	1.11	295	5.72	1.13
5	8	10.61	-4.32%	134	3.42	1.01	193	3.76	1.18	273	10.4	1.22
4	8	10.11	-8.84%	130	2.77	1.04	185	13.6	1.23	245	6.45	1.36
2	8	8.59	-22.48%	126	4.77	1.07	163	6	1.39	203	4.1	1.64
1	8	7.70	-30.57%	126	3.38	1.07	148	2.02	1.53	180	2.85	1.84
6	6	10.73	-3.24%	104	0.45	1.30	178	3.24	1.28	254	2.37	1.31
5	5	10.19	-8.04%	91.6	2.10	1.47	151	1.78	1.50	219	10.3	1.52
4	4	9.50	-14.31%	79	3.38	1.71	124	2.42	1.83	178	1.59	1.87
2	2	7.31	-34.09%	49.5	2.56	2.73	74.8	1.9	3.03	101	0.719	3.29
1	1	4.00	-63.91%	32	1.25	4.22	48.7	2.11	4.66	61.9	1.81	5.36

Table C.21: Role of model symmetry in inference efficiency on FLAN-T5 small model on the XSUM dataset

and Datasets [80] libraries.

[80]https://github.com/huggingface/datasets

l_{enc}	l_{dec}	R-2	Impact	BS 1	STD	Speedup	BS 8	STD	Speedup	BS 16	STD	Speedup
12	12	15.69	0.00%	205	3.81	1.00	546	8.7	1.00	917	4.72	1.00
12	10	15.27	-2.65%	171	2.79	1.20	508	6.39	1.07	876	3.02	1.05
12	8	14.91	-4.97%	150	1.32	1.37	476	2.82	1.15	830	1.08	1.10
12	6	15.40	-1.83%	129	4.33	1.59	450	9.33	1.21	789	3.73	1.16
12	4	15.19	-3.18%	101	2.16	2.03	411	5.27	1.33	744	1.71	1.23
12	2	13.73	-12.47%	76	1.76	2.70	380	3.43	1.44	706	8.13	1.30
10	12	15.92	1.47%	200	6.37	1.03	494	2.45	1.11	818	1.72	1.12
8	12	14.10	-10.09%	195	5.47	1.05	445	20.8	1.23	713	1.71	1.29
6	12	13.84	-11.79%	190	3.89	1.08	396	9.79	1.38	612	4.64	1.50
4	12	12.10	-22.88%	185	2.24	1.11	337	3.09	1.62	505	1.96	1.82
2	12	10.27	-34.53%	180	2.08	1.14	282	4.03	1.94	399	2.85	2.30
10	10	15.72	0.22%	174	4.09	1.18	475	18.5	1.15	772	1.79	1.19
8	8	14.30	-8.85%	140	1.95	1.46	373	2.21	1.46	625	1.51	1.47
6	6	12.44	-20.68%	112	1.71	1.83	290	6.77	1.88	480	3.5	1.91
4	4	10.67	-31.95%	84.2	3.75	2.43	201	1.58	2.72	330	4.43	2.78
2	2	7.74	-50.65%	51.5	3.01	3.98	112	1.02	4.88	179	0.894	5.12

Table C.22: Role of model symmetry in inference efficiency on FLAN-T5 base model on the XSUM dataset

l_{enc}	l_{dec}	R-2	Impact	BS 1	STD	Speedup	BS 8	STD	Speedup	BS 16	STD	Speedup
24	24	16.34	0.00%	447	19.4	1.00	1480	23	1.00	2700	16.1	1.00
24	20	19.80	21.16%	374	4.84	1.20	1410	17.5	1.05	2580	7.52	1.05
24	16	19.30	18.09%	327	19.4	1.37	1320	8.18	1.12	2460	7.19	1.10
24	12	18.92	15.77%	272	7.91	1.64	1240	7.06	1.19	2340	7.5	1.15
24	8	17.96	9.93%	216	7.81	2.07	1170	11.4	1.26	2210	6.49	1.22
24	4	16.47	0.76%	165	3.11	2.71	1090	3.66	1.36	2080	7.17	1.30
20	24	19.43	18.88%	406	21.5	1.10	1310	11.5	1.13	2390	7.76	1.13
16	24	18.33	12.16%	412	20.3	1.08	1140	6.88	1.30	2080	7.01	1.30
12	24	16.90	3.39%	384	18.8	1.16	986	11	1.50	1750	686	1.54
8	24	14.97	-8.39%	369	8.87	1.21	822	15.5	1.80	1420	15.5	1.90
4	24	12.52	-23.37%	345	4.41	1.30	649	3.26	2.28	110	5.96	24.55
20	20	19.18	17.38%	357	11.8	1.25	1230	13.2	1.20	2260	2.16	1.19
16	16	17.56	7.43%	288	5.91	1.55	995	9.41	1.49	1820	5.33	1.48
12	12	15.89	-2.79%	217	3.09	2.06	748	3.25	1.98	1370	6.59	1.97
8	8	12.15	-25.66%	158	6.04	2.83	511	9.62	2.90	920	2.06	2.93
4	4	8.96	-45.14%	92.3	2.88	4.84	267	1.51	5.54	481	1.69	5.61

Table C.23: Role of model symmetry in inference efficiency on FLAN-T5 large model on the XSUM dataset

REFERENCES

[1] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Communications of the ACM*, vol. 60, pp. 84–90, 2012.

[2] T. B. Brown, B. Mann, N. Ryder, *et al.*, "Language models are few-shot learners," *ArXiv*, vol. abs/2005.14165, 2020.

[3] W. Fedus, B. Zoph, and N. M. Shazeer, "Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity," *ArXiv*, vol. abs/2101.03961, 2021.

[4] J. Pennington, R. Socher, and C. D. Manning, "Glove: Global vectors for word representation," in *EMNLP*, 2014.

[5] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean, "Distributed representations of words and phrases and their compositionality," *ArXiv*, vol. abs/1310.4546, 2013.

[6] R. Socher, A. Perelygin, J. Wu, *et al.*, "Recursive deep models for semantic compositionality over a sentiment treebank," in *EMNLP*, 2013.

[7] P. Rajpurkar, J. Zhang, K. Lopyrev, and P. Liang, "Squad: 100,000+ questions for machine comprehension of text," in *EMNLP*, 2016.

[8] M. Seo, A. Kembhavi, A. Farhadi, and H. Hajishirzi, "Bidirectional attention flow for machine comprehension," *ArXiv*, vol. abs/1611.01603, 2017.

[9] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," in *NAACL*, 2019.

[10] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, and I. Sutskever, "Language models are unsupervised multitask learners," 2019. [Online]. Available: `https://d4mucfpksywv.cloudfront.net/better-language-models/language-models.pdf`.

[11] C. Raffel, N. M. Shazeer, A. Roberts, *et al.*, "Exploring the limits of transfer learning with a unified text-to-text transformer," *ArXiv*, vol. abs/1910.10683, 2020.

[12] B. Bi, C. Li, C. Wu, M. Yan, and W. Wang, "Palm: Pre-training an autoencoding&autoregressive language model for context-conditioned generation," in *EMNLP*, 2020.

[13] M. Nadeem, A. Bethke, and S. Reddy, "Stereoset: Measuring stereotypical bias in pretrained language models," in *ACL*, 2021.

[14] E. M. Bender, T. Gebru, A. McMillan-Major, and S. Shmitchell, "On the dangers of stochastic parrots: Can language models be too big?" *Proceedings of the 2021 ACM Conference on Fairness, Accountability, and Transparency*, 2021.

[15] F. Petroni, T. Rocktäschel, P. Lewis, *et al.*, "Language models as knowledge bases?" *ArXiv*, vol. abs/1909.01066, 2019.

[16] Z. Jiang, F. F. Xu, J. Araki, and G. Neubig, "How can we know what language models know?" *Transactions of the Association for Computational Linguistics*, vol. 8, pp. 423–438, 2020.

[17] E. Strubell, A. Ganesh, and A. McCallum, "Energy and policy considerations for modern deep learning research," in *AAAI*, 2020.

[18] E. Strubell, A. Ganesh, and A. McCallum, "Energy and policy considerations for deep learning in nlp," *ArXiv*, vol. abs/1906.02243, 2019.

[19] J. G. M. FitzGerald, S. Ananthakrishnan, K. Arkoudas, *et al.*, "Alexa teacher model: Pretraining and distilling multi-billion-parameter encoders for natural language understanding systems," *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, 2022.

[20] J. Hestness, S. Narang, N. Ardalani, *et al.*, "Deep learning scaling is predictable, empirically," *ArXiv*, vol. abs/1712.00409, 2017.

[21] D. Hernandez, J. Kaplan, T. J. Henighan, and S. McCandlish, "Scaling laws for transfer," *ArXiv*, vol. abs/2102.01293, 2021.

[22] J. Kaplan, S. McCandlish, T. J. Henighan, *et al.*, "Scaling laws for neural language models," *ArXiv*, vol. abs/2001.08361, 2020.

[23] J. Frankle and M. Carbin, "The lottery ticket hypothesis: Finding sparse, trainable neural networks," *arXiv: Learning*, 2019.

[24] T. Chen, J. Frankle, S. Chang, *et al.*, "The lottery ticket hypothesis for pre-trained bert networks," *ArXiv*, vol. abs/2007.12223, 2020.

[25] E. Kurtic, D. F. Campos, T. Nguyen, *et al.*, "The optimal bert surgeon: Scalable and accurate second-order pruning for large language models," *ArXiv*, vol. abs/2203.07259, 2022.

[26] D. F. Campos, A. Marques, T. A. D. Nguyen, M. Kurtz, and C. Zhai, "Sparse*bert: Sparse models are robust," *ArXiv*, vol. abs/2205.12452, 2022.

[27] D. F. Campos, A. Marques, M. Kurtz, and C. Zhai, "Oberta: Improving sparse transfer learning via improved initialization, distillation, and pruning regimes," *ArXiv*, vol. abs/2303.17612, 2023.

[28] V. Karpukhin, B. Oğuz, S. Min, *et al.*, "Dense passage retrieval for open-domain question answering," *ArXiv*, vol. abs/2004.04906, 2020.

[29] A. Magnani, F. Liu, S. Chaidaroon, *et al.*, "Semantic retrieval at walmart," *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, 2022.

[30] K. Bi, Q. Ai, and W. B. Croft, "A transformer-based embedding model for personalized product search," *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval*, 2020.

[31] Y. Qu, Y. Ding, J. Liu, *et al.*, "Rocketqa: An optimized training approach to dense passage retrieval for open-domain question answering," in *NAACL*, 2021.

[32] L. Xiong, C. Xiong, Y. Li, *et al.*, "Approximate nearest neighbor negative contrastive learning for dense text retrieval," *ArXiv*, vol. abs/2007.00808, 2021.

[33] G. Sidiropoulos and E. Kanoulas, "Analysing the robustness of dual encoders for dense retrieval against misspellings," *Proceedings of the 45th International ACM SIGIR Conference on Research and Development in Information Retrieval*, 2022.

[34] D. F. Campos, C. Zhai, and A. Magnani, "Noise-robust dense retrieval via contrastive alignment post training," *ArXiv*, vol. abs/2304.03401, 2023.

[35] D. F. Campos, A. Magnani, and C. Zhai, "Quick dense retrievers consume kale: Post training kullback leibler alignment of embeddings for asymmetrical dual encoders," *ArXiv*, vol. abs/2304.01016, 2023.

[36] A. Vaswani, N. M. Shazeer, N. Parmar, *et al.*, "Attention is all you need," in *NIPS*, 2017.

[37] J. Zhang, Y. Zhao, M. Saleh, and P. J. Liu, "Pegasus: Pre-training with extracted gap-sentences for abstractive summarization," *ArXiv*, vol. abs/1912.08777, 2020.

[38] A. Radford, J. W. Kim, T. Xu, G. Brockman, C. McLeavey, and I. Sutskever, "Robust speech recognition via large-scale weak supervision," *ArXiv*, vol. abs/2212.04356, 2022.

[39] C. Z. Daniel Campos, "To asymmetry and beyond: Structured pruning of sequence to sequence models for improved inference efficiency," *ArXiv*, vol. abs/2304.02721, 2023.

[40] S. Narayan, S. B. Cohen, and M. Lapata, "Don't give me the details, just the summary! topic-aware convolutional neural networks for extreme summarization," *ArXiv*, vol. abs/1808.08745, 2018.

[41] R. Nallapati, B. Zhou, C. N. dos Santos, Ç. Gülçehre, and B. Xiang, "Abstractive text summarization using sequence-to-sequence rnns and beyond," in *CoNLL*, 2016.

[42] T. Isbister, F. Carlsson, and M. Sahlgren, "Should we stop training more monolingual models, and simply use machine translation instead?" In *NODALIDA*, 2021.

[43] A. Conneau, K. Khandelwal, N. Goyal, *et al.*, "Unsupervised cross-lingual representation learning at scale," in *ACL*, 2020.

[44] D. Campos, D. Perry, S. Joshi, *et al.*, "Compressing cross-lingual multi-task models at qualtrics," *ArXiv*, vol. abs/2211.15927, 2022.

[45] M. E. Peters, M. Neumann, M. Iyyer, *et al.*, "Deep contextualized word representations," *ArXiv*, vol. abs/1802.05365, 2018.

[46] K. Clark, M.-T. Luong, Q. V. Le, and C. D. Manning, "Electra: Pre-training text encoders as discriminators rather than generators," *ArXiv*, vol. abs/2003.10555, 2020.

[47] J. Howard and S. Ruder, "Universal language model fine-tuning for text classification," in *ACL*, 2018.

[48] A. Radford, K. Narasimhan, T. Salimans, and I. Sutskever, "Improving language understanding by generative pre-training," 2018. [Online]. Available: `https://cdn.openai.com/research-covers/language-unsupervised/language_understanding_paper.pdf`.

[49] Y. Wu, M. Schuster, Z. Chen, *et al.*, "Google's neural machine translation system: Bridging the gap between human and machine translation," *ArXiv*, vol. abs/1609.08144, 2016.

[50] W. L. Taylor, ""cloze procedure": A new tool for measuring readability," *Journalism Mass Communication Quarterly*, vol. 30, pp. 415–433, 1953.

[51] Y. Liu, M. Ott, N. Goyal, *et al.*, "Roberta: A robustly optimized bert pretraining approach," *ArXiv*, vol. abs/1907.11692, 2019.

[52] Z. Yang, Z. Dai, Y. Yang, J. G. Carbonell, R. Salakhutdinov, and Q. V. Le, "Xlnet: Generalized autoregressive pretraining for language understanding," in *NeurIPS*, 2019.

[53] Z. Dai, Z. Yang, Y. Yang, J. Carbonell, Q. V. Le, and R. Salakhutdinov, "Transformer-xl: Attentive language models beyond a fixed," *ArXiv*, vol. abs/1901.02860, 2019.

[54] Z.-Z. Lan, M. Chen, S. Goodman, K. Gimpel, P. Sharma, and R. Soricut, "Albert: A lite bert for self-supervised learning of language representations," *ArXiv*, vol. abs/1909.11942, 2019.

[55] V. Sanh, L. Debut, J. Chaumond, and T. Wolf, "Distilbert, a distilled version of bert: Smaller, faster, cheaper and lighter," *ArXiv*, vol. abs/1910.01108, 2019.

[56] F. Yu, D. Wang, L. Shangguan, *et al.*, "A survey of large-scale deep learning serving system optimization: Challenges and opportunities," *ArXiv*, vol. abs/2111.14247, 2021.

[57] S. Han, H. Mao, and W. J. Dally, "A deep neural network compression pipeline: Pruning, quantization, huffman encoding," *ArXiv*, 2015.

[58] Y. LeCun, J. S. Denker, and S. A. Solla, "Optimal brain damage," in *NIPS*, 1989.

[59] S. Han, H. Mao, and W. Dally, "Deep compression: Compressing deep neural network with pruning, trained quantization and huffman coding," *arXiv: Computer Vision and Pattern Recognition*, 2016.

[60] O. Kovaleva, S. Kulshreshtha, A. Rogers, and A. Rumshisky, "BERT busters: Outlier layernorm dimensions that disrupt BERT," *CoRR*, vol. abs/2105.06990, 2021. arXiv: `2105.06990`. [Online]. Available: `https://arxiv.org/abs/2105.06990`.

[61] P. Molchanov, S. Tyree, T. Karras, T. Aila, and J. Kautz, "Pruning convolutional neural networks for resource efficient inference," *arXiv: Learning*, 2017.

[62] T. Chen, B. Ji, T. Ding, *et al.*, "Only train once: A one-shot neural network training and pruning framework," in *Neural Information Processing Systems*, 2021.

[63] L. Miao, X. Luo, T. Chen, W. Chen, D. Liu, and Z. Wang, "Learning pruning-friendly networks via frank-wolfe: One-shot, any-sparsity, and no retraining," in *ICLR*, 2022.

[64] N. Lee, T. Ajanthan, and P. H. S. Torr, "Snip: Single-shot network pruning based on connection sensitivity," *ArXiv*, vol. abs/1810.02340, 2019.

[65] T. Gale, E. Elsen, and S. Hooker, "The state of sparsity in deep neural networks," *ArXiv*, vol. abs/1902.09574, 2019.

[66] V. Sanh, T. Wolf, and A. M. Rush, "Movement pruning: Adaptive sparsity by fine-tuning," *ArXiv*, vol. abs/2005.07683, 2020.

[67] B. Hassibi and D. G. Stork, *Second order derivatives for network pruning: Optimal brain surgeon*. Morgan Kaufmann, 1993.

[68] S. P. Singh and D. Alistarh, "Woodfisher: Efficient second-order approximation for neural network compression," *Advances in Neural Information Processing Systems*, vol. 33, 2020.

[69] O. Zafrir, A. Larey, G. Boudoukh, H. Shen, and M. Wasserblat, *Prune once for all: Sparse pre-trained language models*, 2021. arXiv: `2111.05754 [cs.CL]`.

[70] P. Michel, O. Levy, and G. Neubig, "Are sixteen heads really better than one?" In *NeurIPS*, 2019.

[71] E. Voita, D. Talbot, F. Moiseev, R. Sennrich, and I. Titov, "Analyzing multi-head self-attention: Specialized heads do the heavy lifting, the rest can be pruned," in *ACL*, 2019.

[72] S. N. Sridhar and A. Sarah, "Undivided attention: Are intermediate layers necessary for bert?" *ArXiv*, vol. abs/2012.11881, 2020.

[73] H. Sajjad, F. Dalvi, N. Durrani, and P. Nakov, "Poor man's BERT: smaller and faster transformer models," *CoRR*, vol. abs/2004.03844, 2020. arXiv: `2004.03844`. [Online]. Available: `https://arxiv.org/abs/2004.03844`.

[74] A. de Wynter and D. J. Perry, "Optimal subarchitecture extraction for BERT," *CoRR*, vol. abs/2010.10499, 2020. arXiv: `2010.10499`. [Online]. Available: `https://arxiv.org/abs/2010.10499`.

[75] F. Lagunas, E. Charlaix, V. Sanh, and A. Rush, "Block pruning for faster transformers," in *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, Online and Punta Cana, Dominican Republic: Association for Computational Linguistics, Nov. 2021, pp. 10 619–10 629. DOI: `10.18653/v1/2021.emnlp-main.829`. [Online]. Available: `https://aclanthology.org/2021.emnlp-main.829`.

[76] G. E. Hinton, O. Vinyals, and J. Dean, "Distilling the knowledge in a neural network," *ArXiv*, vol. abs/1503.02531, 2015.

[77] X. Jiao, Y. Yin, L. Shang, *et al.*, "Tinybert: Distilling bert for natural language understanding," *ArXiv*, vol. abs/1909.10351, 2020.

[78] Z. Sun, H. Yu, X. Song, R. Liu, Y. Yang, and D. Zhou, "Mobilebert: A compact task-agnostic bert for resource-limited devices," in *ACL*, 2020.

[79] W. Wang, F. Wei, L. Dong, H. Bao, N. Yang, and M. Zhou, "Minilm: Deep self-attention distillation for task-agnostic compression of pre-trained transformers," *ArXiv*, vol. abs/2002.10957, 2020.

[80] M. Courbariaux, I. Hubara, D. Soudry, R. El-Yaniv, and Y. Bengio, "Binarized neural networks: Training deep neural networks with weights and activations constrained to +1 or -1," *arXiv: Learning*, 2016.

[81] O. Zafrir, G. Boudoukh, P. Izsak, and M. Wasserblat, "Q8bert: Quantized 8bit bert," *2019 Fifth Workshop on Energy Efficient Machine Learning and Cognitive Computing - NeurIPS Edition (EMC2-NIPS)*, pp. 36–39, 2019.

[82] W. Zhang, L. Hou, Y. Yin, *et al.*, "Ternarybert: Distillation-aware ultra-low bit bert," *arXiv preprint arXiv:2009.12812*, 2020.

[83] A. Fan, P. Stock, B. Graham, *et al.*, "Training with quantization noise for extreme model compression," *arXiv preprint arXiv:2004.07320*, 2020.

[84] J. Johnson, M. Douze, and H. Jégou, "Billion-scale similarity search with GPUs," *IEEE Transactions on Big Data*, vol. 7, no. 3, pp. 535–547, 2019.

[85] N. Reimers and I. Gurevych, "Sentence-bert: Sentence embeddings using siamese bert-networks," *ArXiv*, vol. abs/1908.10084, 2019.

[86] T. Mesquita, B. Martins, and M. Almeida, "Dense template retrieval for customer support," in *COLING*, 2022.

[87] S.-C. Lin, J.-H. Yang, and J. J. Lin, "In-batch negatives for knowledge distillation with tightly-coupled teachers for dense retrieval," in *REPL4NLP*, 2021.

[88] M. Li, X. Ma, and J. Lin, "An encoder attribution analysis for dense passage retriever in open-domain question answering," *Proceedings of the 2nd Workshop on Trustworthy Natural Language Processing (TrustNLP 2022)*, 2022.

[89] N. Thakur, N. Reimers, A. Ruckl'e, A. Srivastava, and I. Gurevych, "Beir: A heterogenous benchmark for zero-shot evaluation of information retrieval models," *ArXiv*, vol. abs/2104.08663, 2021.

[90] A. Mikołajczyk and M. Grochowski, "Data augmentation for improving deep learning in image classification problem," *2018 International Interdisciplinary PhD Workshop (IIPhDW)*, pp. 117–122, 2018.

[91] Z. Zhong, L. Zheng, G. Kang, S. Li, and Y. Yang, "Random erasing data augmentation," in *AAAI*, 2020.

[92] Y. Li, X. Li, Y. Yang, and R. Dong, "A diverse data augmentation strategy for low-resource neural machine translation," *Inf.*, vol. 11, p. 255, 2020.

[93] K. Lu, P. Mardziel, F. Wu, P. Amancharla, and A. Datta, "Gender bias in neural natural language processing," in *Logic, Language, and Security*, 2020.

[94] S. Y. Feng, V. Gangal, J. Wei, *et al.*, "A survey of data augmentation approaches for nlp," *ArXiv*, vol. abs/2105.03075, 2021.

[95] S. Chopra, R. Hadsell, and Y. LeCun, "Learning a similarity metric discriminatively, with application to face verification," *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, vol. 1, 539–546 vol. 1, 2005.

[96] F. Schroff, D. Kalenichenko, and J. Philbin, "Facenet: A unified embedding for face recognition and clustering," *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 815–823, 2015.

[97] X. Chen, J. Luo, B. He, L. Sun, and Y. Sun, "Towards robust dense retrieval via local ranking alignment," in *International Joint Conference on Artificial Intelligence*, 2022.

[98] I. Yamada, A. Asai, and H. Hajishirzi, "Efficient passage retrieval with hashing for open-domain question answering," *ArXiv*, vol. abs/2106.00882, 2021.

[99] S. Min, J. L. Boyd-Graber, C. Alberti, *et al.*, "Neurips 2020 efficientqa competition: Systems, analyses and lessons learned," in *NeurIPS*, 2020.

[100] N. Thakur, N. Reimers, and J. Lin, "Domain adaptation for memory-efficient dense retrieval," *ArXiv*, vol. abs/2205.11498, 2022.

[101] J. Choi, E. Jung, J. Suh, and W. Rhee, "Improving bi-encoder document ranking models with two rankers and multi-teacher distillation," *Proceedings of the 44th International ACM SIGIR Conference on Research and Development in Information Retrieval*, 2021.

[102] J. Hoffmann, S. Borgeaud, A. Mensch, *et al.*, "Training compute-optimal large language models," *ArXiv*, vol. abs/2203.15556, 2022.

[103] Z. Li, E. Wallace, S. Shen, *et al.*, "Train large, then compress: Rethinking model size for efficient training and inference of transformers," *ArXiv*, vol. abs/2002.11794, 2020.

[104] C. Na, S. V. Mehta, and E. Strubell, "Train flat, then compress: Sharpness-aware minimization learns more compressible models," *ArXiv*, vol. abs/2205.12694, 2022.

[105] J. S. Rosenfeld, J. Frankle, M. Carbin, and N. Shavit, "On the predictability of pruning across scales," *ArXiv*, vol. abs/2006.10621, 2020.

[106] M. Ranzato, S. Chopra, M. Auli, and W. Zaremba, "Sequence level training with recurrent neural networks," *CoRR*, vol. abs/1511.06732, 2015.

[107] T. Mihaylova and A. F. T. Martins, "Scheduled sampling for transformers," in *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics: Student Research Workshop*, Florence, Italy: Association for Computational Linguistics, Jul. 2019, pp. 351–356. DOI: `10.18653/v1/P19-2049`. [Online]. Available: `https://aclanthology.org/P19-2049`.

[108] J. Kasai, N. Pappas, H. Peng, J. Cross, and N. A. Smith, "Deep encoder, shallow decoder: Reevaluating non-autoregressive machine translation," in *International Conference on Learning Representations*, 2020.

[109] Y. Tay, M. Dehghani, J. Rao, *et al.*, "Scale efficiently: Insights from pre-training and fine-tuning transformers," *ArXiv*, vol. abs/2109.10686, 2021.

[110] S. Shleifer and A. Rush, vol. abs/2010.13002, 2020. arXiv: `2010.13002`. [Online]. Available: `https://arxiv.org/abs/2010.13002`.

[111] F. Lagunas, E. Charlaix, V. Sanh, and A. M. Rush, "Block pruning for faster transformers," *ArXiv*, vol. abs/2109.04838, 2021.

[112] Z. Li, Z. Wang, M. Tan, *et al.*, "Dq-bart: Efficient sequence-to-sequence model via joint distillation and quantization," in *Annual Meeting of the Association for Computational Linguistics*, 2022.

[113] T. Schuster, A. Fisch, J. Gupta, *et al.*, "Confident adaptive language modeling," *ArXiv*, vol. abs/2207.07061, 2022.

[114] V. Sanh, L. Debut, J. Chaumond, and T. Wolf, "Distilbert, a distilled version of bert: Smaller, faster, cheaper and lighter," *arXiv preprint arXiv:1910.01108*, 2019.

[115] J. Xin, R. Tang, J. Lee, Y. Yu, and J. J. Lin, "Deebert: Dynamic early exiting for accelerating bert inference," in *ACL*, 2020.

[116] S. Kim, S. Shen, D. Thorsley, A. Gholami, J. Hassoun, and K. Keutzer, "Learned token pruning for transformers," *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, 2021.

[117] *Using deepspeed and megatron to train megatron-turing nlg 530b, the world's largest and most powerful generative language model*, `https://bit.ly/3DlbPIF/`, Accessed: 2021-11-09.

[118] D. Xu, I. E.-H. Yen, J. Zhao, and Z. Xiao, "Rethinking network pruning – under the pre-train and fine-tune paradigm," in *NAACL*, 2021.

[119] NeuralMagic, *Deep sparse: A fast cpu inference engine*, 2021. eprint: `https://github.com/neuralmagic/deepsparse`.

[120] E. Frantar, E. Kurtic, and D. Alistarh, "M-fac: Efficient matrix-free approximations of second-order information," *Advances in Neural Information Processing Systems*, vol. 34, 2021.

[121] S. Shen, Z. Dong, J. Ye, *et al.*, "Q-bert: Hessian based ultra low precision quantization of bert," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 34, 2020, pp. 8815–8821.

[122] S. Yu, Z. Yao, A. Gholami, *et al.*, "Hessian-aware pruning and optimal neural implant," in *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, 2022, pp. 3880–3891.

[123] T. Wolf, L. Debut, V. Sanh, *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*, Online: Association for Computational Linguistics, Oct. 2020, pp. 38–45. [Online]. Available: `https://www.aclweb.org/anthology/2020.emnlp-demos.6`.

[124] M. Kurtz, J. Kopinsky, R. Gelashvili, *et al.*, "Inducing and exploiting activation sparsity for fast inference on deep neural networks," in *Proceedings of the 37th International Conference on Machine Learning*, H. D. III and A. Singh, Eds., ser. Proceedings of Machine Learning Research, vol. 119, Virtual: PMLR, 13–18 Jul 2020, pp. 5533–5543. [Online]. Available: `http://proceedings.mlr.press/v119/kurtz20a.html`.

[125] Q. Lhoest, A. Villanova del Moral, Y. Jernite, *et al.*, "Datasets: A community library for natural language processing," in *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, Association for Computational Linguistics, Nov. 2021, pp. 175–184.

[126] Z. Wang, W. Hamza, and R. Florian, "Bilateral multi-perspective matching for natural language sentences," *CoRR*, vol. abs/1702.03814, 2017. arXiv: `1702.03814`. [Online]. Available: `http://arxiv.org/abs/1702.03814`.

[127] A. Williams, N. Nangia, and S. Bowman, "A broad-coverage challenge corpus for sentence understanding through inference," in *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, New Orleans, Louisiana: Association for Computational Linguistics, 2018, pp. 1112–1122. [Online]. Available: `http://aclweb.org/anthology/N18-1101`.

[128] M. Zhu and S. Gupta, "To prune, or not to prune: Exploring the efficacy of pruning for model compression," *ArXiv*, vol. abs/1710.01878, 2018.

[129] I. Turc, M.-W. Chang, K. Lee, and K. Toutanova, "Well-read students learn better: The impact of student initialization," *ArXiv*, vol. abs/1908.08962, 2019.

[130] A. Fan, E. Grave, and A. Joulin, "Reducing transformer depth on demand with structured dropout," *arXiv preprint arXiv:1909.11556*, 2019.

[131] B. Jacob, S. Kligys, B. Chen, *et al.*, "Quantization and training of neural networks for efficient integer-arithmetic-only inference," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 2704–2713.

[132] R. Bommasani, D. A. Hudson, E. Adeli, *et al.*, "On the opportunities and risks of foundation models," *ArXiv*, vol. abs/2108.07258, 2021.

[133] J. Lee, W. Yoon, S. Kim, *et al.*, "Biobert: A pre-trained biomedical language representation model for biomedical text mining," *Bioinformatics*, vol. 36, pp. 1234–1240, 2020.

[134] I. Chalkidis, M. Fergadiotis, P. Malakasiotis, N. Aletras, and I. Androutsopoulos, "Legal-bert: The muppets straight out of law school," *ArXiv*, vol. abs/2010.02559, 2020.

[135] I. Beltagy, K. Lo, and A. Cohan, "Scibert: A pretrained language model for scientific text," in *EMNLP*, 2019.

[136] H. Peng, N. Pappas, D. Yogatama, R. Schwartz, N. A. Smith, and L. Kong, "Random feature attention," *ArXiv*, vol. abs/2103.02143, 2021.

[137] O. Zafrir, A. Larey, G. Boudoukh, H. Shen, and M. Wasserblat, "Prune once for all: Sparse pre-trained language models," *ArXiv*, vol. abs/2111.05754, 2021.

[138] E. Kurtić, D. F. Campos, T. Nguyen, *et al.*, "The optimal bert surgeon: Scalable and accurate second-order pruning for large language models," *ArXiv*, vol. abs/2203.07259, 2022.

[139] C.-F. Chen, Q. Fan, and R. Panda, "Crossvit: Cross-attention multi-scale vision transformer for image classification," *2021 IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 347–356, 2021.

[140] A. Arnab, M. Dehghani, G. Heigold, C. Sun, M. Lucic, and C. Schmid, "Vivit: A video vision transformer," *2021 IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 6816–6826, 2021.

[141] O. Kovaleva, S. Kulshreshtha, A. Rogers, and A. Rumshisky, "Bert busters: Outlier dimensions that disrupt transformers," in *FINDINGS*, 2021.

[142] M. A. Gordon, K. Duh, and N. Andrews, "Compressing bert: Studying the effects of weight pruning on transfer learning," in *Workshop on Representation Learning for NLP*, 2020.

[143] W. Foundation, *Wikimedia downloads*, 2021. [Online]. Available: `https://dumps.wikimedia.org`.

[144] Y. Zhu, R. Kiros, R. Zemel, *et al.*, "Aligning books and movies: Towards story-like visual explanations by watching movies and reading books," in *The IEEE International Conference on Computer Vision (ICCV)*, Dec. 2015.

[145] Y. Gu, R. Tinn, H. Cheng, *et al.*, "Domain-specific language model pretraining for biomedical natural language processing," *ACM Transactions on Computing for Healthcare (HEALTH)*, vol. 3, pp. 1–23, 2022.

[146] K. R. Kanakarajan, B. Kundumani, and M. Sankarasubbu, "Bioelectra:pretrained biomedical text encoder using discriminators," in *BIONLP*, 2021.

[147] L. L. Smith, L. K. Tanabe, R. J. nee Ando, *et al.*, "Overview of biocreative ii gene mention recognition," *Genome Biology*, vol. 9, S2–S2, 2008.

[148] J. Li, Y. Sun, R. J. Johnson, *et al.*, "Biocreative v cdr task corpus: A resource for chemical disease relation extraction," *Database: The Journal of Biological Databases and Curation*, vol. 2016, 2016.

[149] N. Collier and J.-D. Kim, "Introduction to the bio-entity recognition task at jnlpba," in *NLPBA/BioNLP*, 2004.

[150] R. I. Dogan, R. Leaman, and Z. Lu, "Ncbi disease corpus: A resource for disease name recognition and concept normalization," *Journal of biomedical informatics*, vol. 47, pp. 1–10, 2014.

[151] O. Taboureau, S. K. Nielsen, K. Audouze, *et al.*, "Chemprot: A disease chemical biology database," *Nucleic Acids Research*, vol. 39, pp. D367–D372, 2011.

[152] M. Herrero-Zazo, I. Segura-Bedmar, P. Martínez, and T. Declerck, "The ddi corpus: An annotated corpus with pharmacological substances and drug-drug interactions," *Journal of biomedical informatics*, vol. 46 5, pp. 914–20, 2013.

[153] K. G. Becker, K. C. Barnes, T. J. Bright, and S. A. Wang, "The genetic association database," *Nature Genetics*, vol. 36, pp. 431–432, 2004.

[154] S. Baker, I. Silins, Y. Guo, *et al.*, "Automatic semantic classification of scientific literature according to the hallmarks of cancer," *Bioinformatics*, vol. 32 3, pp. 432–40, 2016.

[155] Q. Jin, B. Dhingra, Z. Liu, W. W. Cohen, and X. Lu, "Pubmedqa: A dataset for biomedical research question answering," in *EMNLP*, 2019.

[156] P. Rajpurkar, J. Zhang, K. Lopyrev, and P. Liang, "Squad: 100, 000+ questions for machine comprehension of text," in *EMNLP*, 2016.

[157] E. Iofinova, A. Peste, M. Kurtz, and D. Alistarh, "How well do sparse imagenet models transfer?" *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 12 256–12 266, 2021.

[158] H. Pouransari and O. Tuzel, "Least squares binary quantization of neural networks," *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pp. 2986–2996, 2020.

[159] E. Kurtic and D. Alistarh, "Gmp*: Well-tuned global magnitude pruning can outperform most bert-pruning methods," *ArXiv*, vol. abs/2210.06384, 2022.

[160] P. Rajpurkar, R. Jia, and P. Liang, "Know what you don't know: Unanswerable questions for squad," in *Annual Meeting of the Association for Computational Linguistics*, 2018.

[161] R. Socher, A. Perelygin, J. Wu, *et al.*, "Recursive deep models for semantic compositionality over a sentiment treebank," in *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, Seattle, Washington, USA: Association for Computational Linguistics, Oct. 2013, pp. 1631–1642. [Online]. Available: `https://www.aclweb.org/anthology/D13-1170`.

[162] SambitSekhar, *First quora dataset release: Question pairs*, Feb. 2017. [Online]. Available: `https://www.kaggle.com/datasets/sambit7/first-quora-dataset`.

[163] A. L. Maas, R. E. Daly, P. T. Pham, D. Huang, A. Y. Ng, and C. Potts, "Learning word vectors for sentiment analysis," in *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, Portland, Oregon, USA: Association for Computational Linguistics, Jun. 2011, pp. 142–150. [Online]. Available: `http://www.aclweb.org/anthology/P11-1015`.

[164] E. F. Tjong Kim Sang and F. De Meulder, "Introduction to the CoNLL-2003 shared task: Language-independent named entity recognition," in *Proceedings of the Seventh Conference on Natural Language Learning at HLT-NAACL 2003*, 2003, pp. 142–147. [Online]. Available: `https://www.aclweb.org/anthology/W03-0419`.

[165] T. Kwiatkowski, J. Palomaki, O. Redfield, *et al.*, "Natural questions: A benchmark for question answering research," *Transactions of the Association for Computational Linguistics*, vol. 7, pp. 453–466, 2019.

[166] M. Li and J. J. Lin, "Encoder adaptation of dense passage retrieval for open-domain question answering," *ArXiv*, vol. abs/2110.01599, 2021.

[167] L. Gao, X. Ma, J. J. Lin, and J. Callan, "Tevatron: An efficient and flexible toolkit for dense retrieval," *ArXiv*, vol. abs/2203.05765, 2022.

[168] D. F. Campos, T. Nguyen, M. Rosenberg, *et al.*, "Ms marco: A human generated machine reading comprehension dataset," *ArXiv*, vol. abs/1611.09268, 2016.

[169] D. Wadden, S. Lin, K. Lo, *et al.*, "Fact or fiction: Verifying scientific claims," in *EMNLP*, 2020.

[170] M. Joshi, E. Choi, D. S. Weld, and L. Zettlemoyer, "Triviaqa: A large scale distantly supervised challenge dataset for reading comprehension," in *ACL*, 2017.

[171] D. Oh, Y. Kim, H. Lee, H. Huang, and H.-J. Lim, "Don't judge a language model by its last layer: Contrastive learning with layer-wise attention pooling," *ArXiv*, vol. abs/2209.05972, 2022.

[172] T. Nguyen, M. Rosenberg, X. Song, *et al.*, "Ms marco: A human generated machine reading comprehension dataset," *arXiv preprint arXiv:1611.09268*, 2016.

[173] S. Zhuang and G. Zuccon, "Dealing with typos for BERT-based passage retrieval and ranking," in *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, Online and Punta Cana, Dominican Republic: Association for Computational Linguistics, Nov. 2021, pp. 2836–2842. DOI: `10.18653/v1/2021.emnlp-main.225`. [Online]. Available: `https://aclanthology.org/2021.emnlp-main.225`.

[174] O. Khattab and M. A. Zaharia, "Colbert: Efficient and effective passage search via contextualized late interaction over bert," *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval*, 2020.

[175] C. Wu, R. Zhang, J. Guo, Y. Fan, and X. Cheng, "Are neural ranking models robust?" *ACM Transactions on Information Systems (TOIS)*, 2021.

[176] G. Penha, A. Câmara, and C. Hauff, "Evaluating the robustness of retrieval pipelines with query variation generators," in *European Conference on Information Retrieval*, 2021.

[177] G. Sidiropoulos, S. Vakulenko, and E. Kanoulas, "On the impact of speech recognition errors in passage retrieval for spoken question answering," *Proceedings of the 31st ACM International Conference on Information & Knowledge Management*, 2022.

[178] S. Zhuang and G. Zuccon, "Characterbert and self-teaching for improving the robustness of dense retrievers on queries with typos," *Proceedings of the 45th International ACM SIGIR Conference on Research and Development in Information Retrieval*, 2022.

[179] K. Mao, Z. Dou, and H. Qian, "Curriculum contrastive context denoising for few-shot conversational dense retrieval," *Proceedings of the 45th International ACM SIGIR Conference on Research and Development in Information Retrieval*, 2022.

[180] S. Hofstätter, S.-C. Lin, J.-H. Yang, J. J. Lin, and A. Hanbury, "Efficiently teaching an effective dense retriever with balanced topic aware sampling," *Proceedings of the 44th International ACM SIGIR Conference on Research and Development in Information Retrieval*, 2021.

[181] S. Bird, E. Klein, and E. Loper, *Natural language processing with Python: analyzing text with the natural language toolkit.* " O'Reilly Media, Inc.", 2009.

[182] C. D. Fellbaum, "Wordnet : An electronic lexical database," *Language*, vol. 76, p. 706, 2000.

[183] Y. Zhang, J. Baldridge, and L. He, "Paws: Paraphrase adversaries from word scrambling," *ArXiv*, vol. abs/1904.01130, 2019.

[184] G. Klein, Y. Kim, Y. Deng, J. Senellart, and A. Rush, "OpenNMT: Open-source toolkit for neural machine translation," in *Proceedings of ACL 2017, System Demonstrations*, Vancouver, Canada: Association for Computational Linguistics, Jul. 2017, pp. 67–72. [Online]. Available: `https://www.aclweb.org/anthology/P17-4012`.

[185] N. Craswell, D. Campos, B. Mitra, E. Yilmaz, and B. Billerbeck, "Orcas: 18 million clicked query-document pairs for analyzing search," in *CIKM 2020*, 2020.

[186] H. Zhang, X. Song, C. Xiong, *et al.*, "Generic intent representation in web search," *Proceedings of the 42nd International ACM SIGIR Conference on Research and Development in Information Retrieval*, 2019.

[187] A. Wang, A. Singh, J. Michael, F. Hill, O. Levy, and S. Bowman, "Glue: A multi-task benchmark and analysis platform for natural language understanding," in *Proceedings of the 2018 EMNLP Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*, 2018, pp. 353–355.

[188] S. Smith, M. Patwary, B. Norick, *et al.*, "Using deepspeed and megatron to train megatron-turing nlg 530b, a large-scale generative language model," *arXiv preprint arXiv:2201.11990*, 2022.

[189] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of deep bidirectional transformers for language understanding," in *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, Minneapolis, Minnesota: Association for Computational Linguistics, Jun. 2019, pp. 4171–4186. DOI: `10.18653/v1/N19-1423`. [Online]. Available: `https://aclanthology.org/N19-1423`.

[190] W. Wang, F. Wei, L. Dong, H. Bao, N. Yang, and M. Zhou, "Minilm: Deep self-attention distillation for task-agnostic compression of pre-trained transformers," *arXiv preprint arXiv:2002.10957*, 2020.

[191] W. Wang, H. Bao, S. Huang, L. Dong, and F. Wei, "Minilmv2: Multi-head self-attention relation distillation for compressing pretrained transformers," *arXiv preprint arXiv:2012.15828*, 2020.

[192] S. Mukherjee, A. H. Awadallah, and J. Gao, "Xtremedistiltransformers: Task transfer for task-agnostic distillation," *arXiv preprint arXiv:2106.04563*, 2021.

[193] X. Jiao, Y. Yin, L. Shang, *et al.*, "Lightmbert: A simple yet effective method for multilingual bert distillation," *arXiv preprint arXiv:2103.06418*, 2021.

[194] X. Jiao, Y. Yin, L. Shang, *et al.*, "Tinybert: Distilling bert for natural language understanding," in *Findings of the Association for Computational Linguistics: EMNLP 2020*, 2020, pp. 4163–4174.

[195] A. de Wynter and D. J. Perry, "Optimal subarchitecture extraction for bert," *arXiv preprint arXiv:2010.10499*, 2020.

[196] Z. Yang, L. Shou, M. Gong, W. Lin, and D. Jiang, "Model compression with multi-task knowledge distillation for web-scale question answering system," *arXiv preprint arXiv:1904.09636*, 2019.

[197] G. Lample and A. Conneau, "Cross-lingual language model pretraining," *CoRR*, vol. abs/1901.07291, 2019. arXiv: `1901.07291`. [Online]. Available: `http://arxiv.org/abs/1901.07291`.

[198] A. Conneau, K. Khandelwal, N. Goyal, *et al.*, "Unsupervised cross-lingual representation learning at scale," in *Annual Meeting of the Association for Computational Linguistics*, 2019.

[199] Y. Liu, M. Ott, N. Goyal, *et al.*, *Roberta: A robustly optimized bert pretraining approach*, 2019. DOI: `10.48550/ARXIV.1907.11692`. [Online]. Available: `https://arxiv.org/abs/1907.11692`.

[200] Y. Lin, S. Yang, V. Stoyanov, and H. Ji, "A multi-lingual multi-task architecture for low-resource sequence labeling," in *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 2018, pp. 799–809.

[201] P. Vijayaraghavan, S. Vosoughi, and D. Roy, "Twitter demographic classification using deep multi-modal multi-task learning," in *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, 2017, pp. 478–483.

[202] R. He, W. S. Lee, H. T. Ng, and D. Dahlmeier, "An interactive multi-task learning network for end-to-end aspect-based sentiment analysis," *arXiv preprint arXiv:1906.06906*, 2019.

[203] G. Hinton, O. Vinyals, and J. Dean, "Distilling the knowledge in a neural network," *arXiv preprint arXiv:1503.02531*, 2015.

[204] S. Mukherjee and A. H. Awadallah, "Xtremedistil: Multi-stage distillation for massive multilingual models," in *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, 2020, pp. 2221–2234.

[205] O. Zafrir, G. Boudoukh, P. Izsak, and M. Wasserblat, "Q8BERT: quantized 8bit BERT," *CoRR*, vol. abs/1910.06188, 2019. arXiv: `1910.06188`. [Online]. Available: `http://arxiv.org/abs/1910.06188`.

[206] A. Paszke, S. Gross, F. Massa, *et al.*, "Pytorch: An imperative style, high-performance deep learning library," in *Advances in Neural Information Processing Systems 32*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, Eds., Curran Associates, Inc., 2019, pp. 8024–8035. [Online]. Available: `http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf`.

[207] T. Wolf, L. Debut, V. Sanh, *et al.*, "Huggingface's transformers: State-of-the-art natural language processing," *arXiv preprint arXiv:1910.03771*, 2019.

[208] S. P. Singh and D. Alistarh, *Woodfisher: Efficient second-order approximation for neural network compression*, 2020. arXiv: `2004.14340 [cs.LG]`.

[209] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," *CoRR*, vol. abs/1409.3215, 2014. arXiv: `1409.3215`. [Online]. Available: `http://arxiv.org/abs/1409.3215`.

[210] T. J. Henighan, J. Kaplan, M. Katz, *et al.*, "Scaling laws for autoregressive generative modeling," *ArXiv*, vol. abs/2010.14701, 2020.

[211] O. Neumann and C. Gros, "Scaling laws for a multi-agent reinforcement learning model," *ArXiv*, vol. abs/2210.00849, 2022.

[212] M. Lewis, Y. Liu, N. Goyal, *et al.*, "Bart: Denoising sequence-to-sequence pre-training for natural language generation," *ArXiv*, vol. abs/1910.13461, 2019.

[213] R. Nogueira, Z. Jiang, R. Pradeep, and J. Lin, "Document ranking with a pretrained sequence-to-sequence model," in *Findings of the Association for Computational Linguistics: EMNLP 2020*, Online: Association for Computational Linguistics, Nov. 2020, pp. 708–718. DOI: `10.18653/v1/2020.findings-emnlp.63`. [Online]. Available: `https://aclanthology.org/2020.findings-emnlp.63`.

[214] J. Wei, M. Bosma, V. Zhao, *et al.*, "Finetuned language models are zero-shot learners," *ArXiv*, vol. abs/2109.01652, 2021.

[215] A. See, P. J. Liu, and C. D. Manning, "Get to the point: Summarization with pointer-generator networks," in *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, Vancouver, Canada: Association for Computational Linguistics, Jul. 2017, pp. 1073–1083. DOI: `10.18653/v1/P17-1099`. [Online]. Available: `https://www.aclweb.org/anthology/P17-1099`.

[216] C.-Y. Lin, "ROUGE: A package for automatic evaluation of summaries," in *Text Summarization Branches Out*, Barcelona, Spain: Association for Computational Linguistics, Jul. 2004, pp. 74–81. [Online]. Available: `https://aclanthology.org/W04-1013`.

[217] K. Yang, V. Yao, J. DeNero, and D. Klein, "A streaming approach for efficient batched beam search," in *Conference on Empirical Methods in Natural Language Processing*, 2020.

[218] S. Mukherjee, A. Mitra, G. Jawahar, S. Agarwal, H. Palangi, and A. H. Awadallah, "Orca: Progressive learning from complex explanation traces of gpt-4," *ArXiv*, vol. abs/2306.02707, 2023.

[219] T. Formal, B. Piwowarski, and S. Clinchant, "Splade: Sparse lexical and expansion model for first stage ranking," in *Proceedings of the 44th International ACM SIGIR Conference on Research and Development in Information Retrieval*. New York, NY, USA: Association for Computing Machinery, 2021, pp. 2288–2292, ISBN: 9781450380379. [Online]. Available: `https://doi.org/10.1145/3404835.3463098`.

[220] A. Overwijk, C. Xiong, and J. Callan, "Clueweb22: 10 billion web documents with rich information," *Proceedings of the 45th International ACM SIGIR Conference on Research and Development in Information Retrieval*, 2022.

[221] X. Yu, G. Li, C. Chai, and N. Tang, "Reinforcement learning with tree-lstm for join order selection," in *2020 IEEE 36th International Conference on Data Engineering (ICDE)*, 2020, pp. 1297–1308. DOI: `10.1109/ICDE48307.2020.00116`.

[222] R. Rafailov, A. Sharma, E. Mitchell, S. Ermon, C. D. Manning, and C. Finn, "Direct preference optimization: Your language model is secretly a reward model," *ArXiv*, vol. abs/2305.18290, 2023.

[223] A. Ouared and F. Z. Kharroubi, "Moving database cost models from darkness to light," in *Smart Applications and Data Analysis: Third International Conference, SADASC 2020, Marrakesh, Morocco, June 25–26, 2020, Proceedings 3*, Springer, 2020, pp. 17–32.

[224] S. Kim, S. Shen, D. Thorsley, *et al.*, "Learned token pruning for transformers," in *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, ser. KDD '22, Association for Computing Machinery, 2022, pp. 784–794, ISBN: 9781450393850.

[225] L. Weijie, Z. Peng, Z. Zhe, W. Zhiruo, D. Haotang, and J. Qi, "Fastbert: A self-distilling bert with adaptive inference time," in *Proceedings of ACL 2020*, 2020.

[226] Y. Zhu, R. Kiros, R. S. Zemel, *et al.*, "Aligning books and movies: Towards story-like visual explanations by watching movies and reading books," *2015 IEEE International Conference on Computer Vision (ICCV)*, pp. 19–27, 2015.

[227] A. Conneau, K. Khandelwal, N. Goyal, *et al.*, *Unsupervised cross-lingual representation learning at scale*, 2019. DOI: `10.48550/ARXIV.1911.02116`. [Online]. Available: `https://arxiv.org/abs/1911.02116`.