Efficient Scaling of Language Models

Artidoro Pagnoni

A dissertation

submitted in partial fulfillment of the

requirements for the degree of

Doctor of Philosophy

University of Washington

2025

Reading Committee:
Luke Zettlemoyer, Chair
Hannaneh Hajishirzi
Pang Wei Koh

Program Authorized to Offer Degree:
Computer Science and Engineering

University of Washington

Abstract

Efficient Scaling of Language Models

Artidoro Pagnoni

Chair of the Supervisory Committee:
Luke Zettlemoyer
Department of Computer Science and Engineering

Large language models (LLMs) are progressively reshaping how humans interact with information, offering increasingly sophisticated access to knowledge through natural language interfaces and advancing reasoning capabilities across diverse domains. Yet, their impressive gains have hinged on a simple recipe: exponentially increasing resources. Over the past years, computational requirements have increased tenfold annually, training costs now reach billions of dollars, and we are rapidly exhausting the internet's high-quality text. This dissertation addresses a critical question: How can we unlock further capabilities from scale while curbing the exponential growth in compute, data, and energy? I present three complementary innovations across the LLM development pipeline that fundamentally improve scaling efficiency.

Traditional LLMs depend on tokenization—a preprocessing step that introduces biases and inefficiencies. BLT eliminates this bottleneck by learning directly from raw bytes, dynamically grouping them into larger entropy-adaptive patches via lightweight encoder-decoder modules. This tokenizer-free architecture not only improves robustness to noisy and multilingual inputs but achieves up to 50% reduction in inference FLOPs while matching tokenization-based models' performance. Controlled scaling experiments demonstrate that BLT enables a new scaling dimension with improved scaling trends over current approaches.

Beyond model architecture, data quality has a significant impact on model performance. Socratic Pretraining transforms unlabeled documents into richer training signal by masking salient sentences, synthet-

ically generating content questions about missing information, and training models to both pose questions and draft answers. Applied to BART-large, this approach achieves state-of-the-art performance on QMSum and SQuALITY (+1.0 and +0.5 ROUGE-1 over strong baselines), halves labeled data requirements, and improves faithfulness across multiple control interfaces—all with minimal computational overhead.

QLoRA combines novel 4-bit quantization with parameter efficient finetuning, reducing memory requirements of supervised learning by 15× without performance degradation. This efficiency enabled comprehensive instruction-tuning studies revealing that data quality, not quantity, drives downstream performance in post-training. Efficient finetuning of the quantized base model also alleviates quantization errors reducing inference memory requirements while enabling full-precision quality.

These methods collectively demonstrate that sustainable scaling requires rethinking fundamental assumptions at each pipeline stage. BLT's entropy-adaptive computation, Socratic Pretraining's synthetic supervision, and QLoRA's quantized adaptation each extract more capability per unit resource—whether FLOPs, tokens, or memory. Together, they chart a practical roadmap for continued LLM progress within more sustainable computational bounds, proving that smarter algorithms can bend the scaling laws.

Acknowledgements

Thank you to the countless people who supported me throughout my PhD journey. It would be impossible to acknowledge everyone whose impact was profound and whose contributions significantly shaped this thesis.

First and foremost, I am deeply grateful to my advisor, Luke Zettlemoyer, for providing me with the strength and guidance to ask bold questions and pursue ambitious ideas. Your unmatched optimism, creativity, and support have been truly invaluable, and I feel incredibly fortunate to have worked alongside you. Our brainstorming sessions and explorations of wild ideas were among the highlights of my graduate career, and I learned immensely from our interactions.

Thank you to Mike Lewis for being an exceptional role model, keeping me grounded, demonstrating the elegance of simplicity and scale, and ensuring we never missed lunch at noon. Your mentorship and guidance have profoundly influenced me. I am equally thankful to Yulia Tsvetkov for teaching me to critically consider the broader impacts of research and how best to present it. Thanks to Sasha Rush for introducing me to NLP during my undergraduate and revealing the mathematical beauty of deep learning. Thank you to my thesis committee Hanna, Pang Wei, and Nadya for the flexibility and support in completing this thesis. Special thanks to Alberto Sangiovanni for encouraging me to pursue a PhD—you were absolutely right!

My research has greatly benefited from numerous insightful conversations and collaborations. I am particularly grateful to Ari Holtzman and Tim Dettmers. Ari, thank you for joining me on the exciting journey of BLT and instilling the belief that we could achieve it. Tim, thank you for emphasizing the importance of first principles and encouraging me to think beyond immediate steps.

I am also thankful for the opportunities provided through the Meta mentorship program, which allowed me to experience an ideal graduate school environment enhanced by exceptional resources. Special thanks to my friends Melanie, Hunter, Margaret, Weijia, Inna, Carolina, Rulin, Stella, Thomasz, and Thao. Thank you

also to the FAIR collaborators and mentors from whom I have learned so much: Srini, Gargi, Pedro, Ram, John, Chunting, Lily, Sachin, Jason, Omer, and Ben. Additionally, I extend my gratitude to my mentors at Salesforce, Alex, Wojciech, Philippe, and Jason, where part of the thesis work was conducted.

Thank you to my ZLabmates for engaging in challenging and enriching research discussions: Bhargavi, Terra, Victor, Hila, Suchin, Sewon, Joel, Jacquelin, Luiza, Kalyani, Sahil, Sneha, Tong, and Ananya. Thanks also to the Tsvetshop lab for welcoming me at CMU and during my first year at UW, especially Anjalie, Chan, Vidhisha, Han, Sachin, and Rishab. My friends and mentors in Pittsburgh—Felix, Brendon, Sruti, Sid, Lasha, Paul, Rishab, and Bhiksha—thank you for your companionship and guidance. I am equally grateful to my UW officemates, friends, and collaborators for making my university experience both stimulating and enjoyable: Ofir, Sam, Gabriel, Alisa, Ethan, Goncalo, Dave, and Ashish. Special thanks to UW advisors Elise, Anna Karlin, Joe, and Chris for their invaluable support in navigating and maximizing my graduate experience.

My heartfelt thanks to my friends and roommates for always being there, sharing fun times, and exploring the beautiful mountains of the Pacific Northwest with me: Gian Marco, Lancelot, Raphael, Sasha, Christina, Riccardo, Kelly, Lorenzo, Sindi, Filip, Pierre, Lindsay, Jacques, Anna, Evgeniia, Xinya, Zhiyao, Moritz, Galen, Ed, Mathieu, Sri, Tim, Clara, Jean-Rémi, Garrett, Hellen, Hunter, Nadya.

Thank you to my families away from home: Andrew and Carol, for your unwavering warmth and support; Megan and Steven, for welcoming me to the United States and making it feel like home.

To my longtime friends Marco, Mario, Giacomo, and Amine—thank you for being a constant anchor throughout the years.

I am profoundly grateful to my family for their unconditional love and constant encouragement. Thank you to my father for instilling rigor, curiosity, and the belief that nothing is impossible. Thank you to my mother for her boundless love and unwavering support. To my sister, thank you for your constant encouragement and joyful presence in my life. And heartfelt thanks to my grandparents for teaching me the importance of bold choices, always offering their love, support, and encouragement.

Finally, thank you, Eva, for being my partner, friend, and collaborator. This thesis is deeply influenced by our conversations and the many choices we have made together. Your presence and support have been indispensable, and I am endlessly grateful to have you by my side.

DEDICATION

To my family.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

We stand at an inflection point where the centuries-old paradigm of humans searching for information is giving way to one in which information converges upon human intent. Large language models (LLMs) are the catalyst of this transformation. Trained on ever-increasing volumes of data—now reaching hundreds of terabytes spanning the breadth of human knowledge—these rapidly scaling models have demonstrated remarkable capabilities: they can draft complex documents in minutes rather than days, generate functional code from natural language specifications, and provide analysis across domains from medicine to law. The same foundational methods that enable these capabilities have proven equally effective across multimodal systems extending to image and video understanding and generation, speech synthesis and recognition, and even protein structure prediction. Their impact extends beyond professional settings—the same models serve as educational tools, creative collaborators, and multilingual communicators, fundamentally altering how we access and apply knowledge. With already hundreds of millions of monthly active users and tens of billions of dollars in annual revenue, LLMs are reshaping entire sectors of the economy [Eloundou et al., 2024].

The remarkable trajectory of language model capabilities [Devlin et al., 2018; Radford et al., 2019; Dubey et al., 2024] has been recently driven by a deceptively simple principle: scale. By increasing model parameters from millions to hundreds of billions, expanding training datasets from gigabytes to terabytes, and multiplying computational resources by orders of magnitude, we have consistently improved performance and unlocked emergent behaviors [Hoffmann et al., 2022; Kaplan et al., 2020]. However, this scal-

ing paradigm faces fundamental sustainability challenges that threaten to constrain future progress. The computational requirements for training state-of-the-art models have increased by an order of magnitude approximately every year[1], far outpacing hardware improvements. Data availability presents another bottleneck—we are rapidly approaching the limits of high-quality text available on the internet, with estimates suggesting we may exhaust unique training data within this decade [Villalobos et al., 2022]. The economic burden is equally daunting, with training costs for frontier models reaching billions of dollars and inference serving billions of requests requiring massive datacenter investments [Pilz et al., 2025]. These exponential resource requirements cannot continue indefinitely. Therefore, the critical research challenge facing our field is not merely how to scale further but how can we achieve greater capability per unit of computational resource across the entire LLM development pipeline without sacrificing performance?

Improving the efficiency of machine learning systems often entails making computational approximations, which suggests a trade-off between performance and cost. Yet, within the scaling paradigm, more efficient yet imprecise systems may be desirable because they enable the use of larger models within the same resource constraints. If the impact of improving the efficiency can be minimized, the larger scale of models could justify the price of these approximations. This dissertation therefore asks:

- How can architectural innovations that adapt computation to data complexity fundamentally improve the scaling efficiency of language models?
- What role does training signal quality play in reducing data requirements, and how can we systematically enhance it through synthetic augmentation?
- Can we decouple model capability from memory requirements through precision reduction techniques that maintain full performance?

At the heart of modern language models lies the Transformer architecture, whose self-attention layers efficiently capture long-range dependencies and have become the de-facto backbone for state-of-the-art systems across modalities. During pre-training, a Transformer is exposed to trillions of text tokens drawn from broad, mostly uncurated sources; its objective is to minimize cross-entropy loss between the model's next-token distribution and the discrete ground-truth sequence, thereby acquiring a general-purpose prior

[1]Llama 3 required more than 30 times the FLOPs used to train Llama 2 and came one year later

over natural language. Mid-training (sometimes called continued pre-training or domain adaptation) refines this prior on higher-quality, task-targeted, or modality-balanced data, tightening the alignment between the model's internal representations and the statistical structure most relevant to downstream use. Finally, post-training techniques—such as instruction tuning, reinforcement learning from human or automated feedback align model behaviors with explicit user goals and refine model performance. Although objectives for other modalities like images, audio, or proteins may swap discrete token prediction for continuous reconstruction, contrastive alignment or diffusion denoising the architectural backbone and main scaling challenges remain largely shared. This thesis therefore focuses on the text modality while drawing on multimodal parallels. Once trained, inference operates through autoregressive generation, where the model produces tokens sequentially, using its own outputs as inputs for subsequent predictions. Recent advances in inference-time scaling have demonstrated that additional computation during generation—through techniques like chain-of-thought reasoning, tree search, or iterative refinement—can significantly improve output quality. Each stage of the LLM pipeline—pretraining, midtraining, post-training, and inference—offers distinct opportunities to translate computational resources into performance gains. This thesis explores practical approaches to improve efficiency across all these stages.

1.1 Architecture and Pretraining–Byte Latent Transformer: Patches Scale Better Than Tokens

chapter 2 introduces the Byte Latent Transformer (**BLT**), a tokenizer-free architecture that fundamentally rethinks how language models process text during pretraining. Unlike traditional LLMs that rely on tokenization–a heuristic pre-processing step that segments text into sequence of tokens from a static vocabulary–BLT operates directly on raw bytes, dynamically grouping them into patches based on prediction complexity. This entropy-adaptive approach allocates more compute to challenging predictions and less to predictable continuations, achieving up to 50% reduction in inference FLOPs without sacrificing performance. BLT's dynamic patching also creates a new scaling dimension by simultaneously increasing both model and patch size within a fixed inference budget. Controlled experiments demonstrate that this architecture not only matches the performance of state-of-the-art tokenization-based models like Llama 3

at 8B scale, but exhibits superior scaling trends—with larger patches becoming increasingly advantageous at greater model scales. Moreover, by preserving byte-level information, BLT demonstrates remarkable robustness to noise and superior performance on character-manipulation tasks where tokenization-based models struggle.

1.2 Data Quality and Mid Training–Socratic Pretraining: Question Driven Pretraining for Controllable Summarization

Socratic Pretraining, described in chapter 3, tackles the mid-training stage by injecting task-specific structure into otherwise unlabelled web corpora. It masks salient sentences in each document, automatically generates sentence-level "content questions" about the missing information, and then trains the model to (i) pose those questions and (ii) draft a pseudo-summary that answers them. This simple question-driven objective helps tune the model to the same fine-grained query/response patterns that controllable summarization requires. This method needs no human annotations beyond a question-generation system. When applied as a light-weight continued pretraining step to BART-LARGE, Socratic Pretraining delivers state-of-the-art performance on the long-document benchmarks QMSum and SQuALITY (≈,+1.0 and +0.5 ROUGE-1 over a strong SegEnc baseline), halves the amount of task-specific labelled data needed, and improves faithfulness across multiple control strategies—including content questions, keyword chains, and QA blueprints. In the broader thesis agenda, it illustrates how judicious, task-aware mid-training can unlock substantial capability gains without resorting to large amounts of expensive human authored datasets or larger models.

1.3 Post-Training and Inference–QLoRA: Efficient Finetuning of Quantized LLMs

Chapter 4 presents **QLoRA**, which revolutionizes instruction-tuning in post-training by reducing memory requirements of finetuning without performance degradation. While traditional finetuning of large models demands prohibitive resources—LLaMA 65B requires over 780 GB of GPU memory—QLoRA reduces this to under 48 GB, a 15× reduction that fits on a single consumer GPU. The key innovation combines three techniques: (1) 4-bit NormalFloat quantization, an information-theoretically optimal data type for normally

distributed weights, (2) Double Quantization, which quantizes the quantization constants themselves for additional memory savings, and (3) Paged Optimizers to handle gradient checkpointing spikes. By backpropagating gradients through frozen 4-bit weights to update only small Low-rank Adapters, QLoRA matches full 16-bit finetuning performance. This efficiency breakthrough enabled an extensive study of instruction tuning recipies–over 1,000 models trained across 8 datasets–finding that data quality dominates quantity. A 9k sample dataset (OASST1) outperformed a 450k sample dataset (FLAN v2) on chatbot benchmarks, challenging assumptions about scaling data in instruction tuning. The resulting Guanaco model achieved 99.3% of ChatGPT's performance on the Vicuna benchmark with 24h of training on consumer hardware, democratizing access to state-of-the-art language model capabilities.

1.4 Summary

This thesis charts a practical roadmap for efficient scaling of large language models by tightening each stage of the development pipeline: (i) in pre-training and architecture, the Byte Latent Transformer replaces token vocabularies with larger entropy-adaptive patches of bytes, cutting inference FLOPs and improving pretraining scaling trends; (ii) during mid-training, where data quality dominates, Socratic Pretraining transforms unlabelled corpora into question-answer supervision, extracting richer signal per token and boosting controllable-summarization performance without larger models or datasets; and (iii) for post-training and inference, QLoRA combines 4-bit quantization with low-rank adapters, reducing supervised fine-tuning memory by a factor of fifteen and enabling commodity-GPU deployment at full-precision quality. Collectively, these contributions raise the performance-per-FLOP curve across architecture, data, and inference, demonstrating that greater capability need not demand unsustainable growth in compute, data, or energy.

Chapter 2

Byte Latent Transformer: Patches Scale Better Than Tokens

In this chapter, we present the Byte Latent Transformer (**BLT**), a tokenizer-free architecture that operates on raw byte data and, for the first time, matches the performance of tokenization-based models, with significant improvements in scaling efficiency of LLM pretraining. This architecture challenges the conventional idea that language models *learn* and *operate* on a single shared representation space. BLT receives rich training signal from the raw byte sequence with cross-entropy loss, while operating on a compressed sequence of latent representations of patches, or groups of bytes. By forgoing the constraints of fixed-vocabulary tokenization, BLT gains the flexibility to allocates patches and compute based on the complexity of the data it is predicting. Ultimately, this enables the model to operate on compressed representations with gains in scaling efficiency while also improving robustness to noise. This chapter includes materials originally published in Pagnoni et al. [2024] (to appear at ACL 2025).

2.1 Introduction

We introduce the Byte Latent Transformer (BLT), a tokenizer-free architecture that learns from raw byte data and, for the first time, matches the performance of tokenization-based models at scale, with significant improvements in efficiency and robustness (§2.6). Existing large language models (LLMs) are trained almost

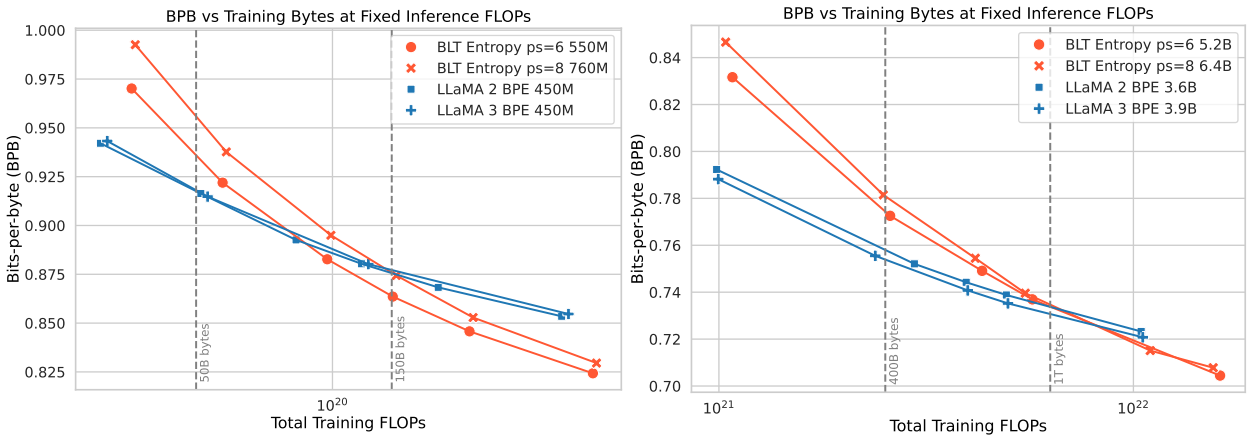

Figure 2.1: Scaling trends for fixed inference FLOP models (fully) trained with varying training budgets. In token-based models, a fixed inference budget determines the model size. In contrast, the BLT architecture provides a new scaling axis allowing simultaneous increases in model and patch size while keeping the same training and inference budget. BLT patch-size (ps) 6 and 8 models quickly overtake scaling trends of BPE Llama 2 and 3. Moving to the larger inference budget makes the larger patch size 8 model more desirable sooner. Both BPE compute-optimal point and crossover point are indicated with vertical lines.

entirely end-to-end, except for tokenization—a heuristic pre-processing step that groups bytes into a static set of tokens. Such tokens bias how a string is compressed, leading to shortcomings such as domain/modality sensitivity [Dagan et al., 2024], sensitivity to input noise (§2.6), a lack of orthographic knowledge [Edman et al., 2024], and multilingual inequity [Liang et al., 2023; Petrov et al., 2023; Limisiewicz et al., 2024].

Tokenization has previously been essential because directly training LLMs on bytes is prohibitively costly at scale due to long sequence lengths [Xue et al., 2022]. Prior works mitigate this by employing more efficient self-attention [El Boukkouri et al., 2020; Clark et al., 2022] or attention-free architectures [Wang et al., 2024] (§2.8). However, this primarily helps train *small models*. At scale, the computational cost of a Transformer is dominated by large feed-forward network layers that run on every byte, not the cost of the attention mechanism.

To efficiently allocate compute, we propose a dynamic, learnable method for grouping bytes into *patches* (§2.2) and a new model architecture that mixes byte and patch information. Unlike tokenization, BLT has no fixed vocabulary for patches. Arbitrary groups of bytes are mapped to latent patch representations via light-weight learned encoder and decoder modules. We show that this results in *more* efficient allocation of compute than tokenization-based models.

Tokenization-based LLMs allocate the same amount of compute to every token. This trades efficiency

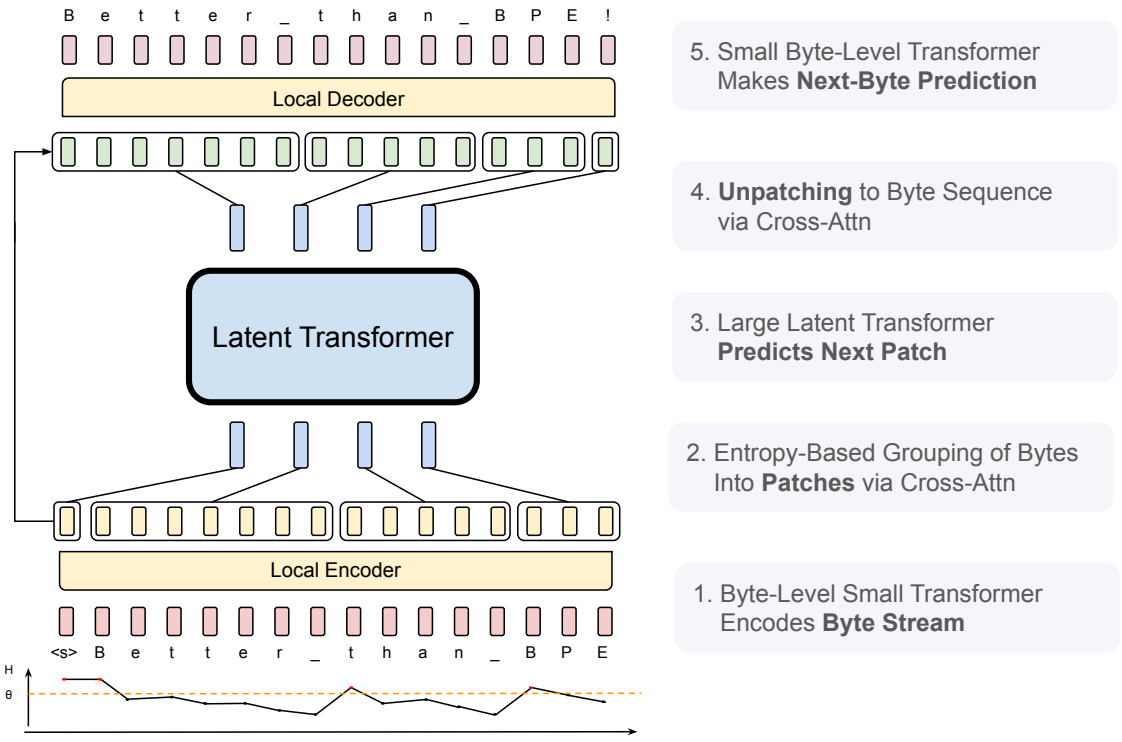

Figure 2.2: BLT comprises three modules, a lightweight *Local Encoder* that encodes input bytes into patch representations, a computationally expensive Latent Transformer over patch representations, and a lightweight *Local Decoder* to decode the next patch of bytes. BLT incorporates byte n-gram embeddings and a cross-attention mechanism to maximize information flow between the Latent Transformer and the byte-level modules (Figure 2.5). Unlike fixed-vocabulary tokenization, BLT dynamically groups bytes into patches preserving access to the byte-level information.

for performance, since tokens are induced with compression heuristics that are not always correlated with the complexity of predictions. Central to our architecture is the idea that models should dynamically allocate compute where it is needed. For example, a large transformer is not needed to predict the ending of most words, since these are comparably easy, low-entropy decisions compared to choosing the first word of a new sentence. This is reflected in BLT's architecture (§2.3) where there are three transformer blocks: two small byte-level *local models* and a large global *latent transformer* (Figure 2.2). To determine how to group bytes into patches and therefore how to dynamically allocate compute, BLT segments data based on the entropy of the next-byte prediction creating contextualized groupings of bytes with relatively uniform information density.

We present the first FLOP-controlled scaling study of byte-level models up to 8B parameters and 4T

training bytes, showing that we can train a model end-to-end at scale from bytes without fixed-vocabulary tokenization. Overall, BLT matches training FLOP-controlled performance[1] of Llama 3 while using up to 50% fewer FLOPs at inference (§2.5). We also show that directly working with raw bytes provides significant improvements in modeling the long-tail of the data. BLT models are more robust than tokenizer-based models to noisy inputs and display enhanced character level understanding abilities demonstrated on orthographic knowledge, phonology, and low-resource machine translation tasks (§2.6). Finally, with BLT models, we can simultaneously increase model size and patch size while maintaining the same inference FLOP budget. Longer patch sizes, on average, save compute which can be reallocated to grow the size of the global latent transformer, because it is run less often. We conduct inference-FLOP controlled scaling experiments (Figure 2.1), and observe significantly better scaling trends than with tokenization-based architectures.

In summary, this chapter makes the following contributions: 1) We introduce BLT, a byte latent LLM architecture that dynamically allocates compute to improve FLOP efficiency, 2) We show that we achieve training FLOP-controlled parity with Llama 3 up to 8B scale while having the option to trade minor losses in evaluation metrics for FLOP efficiency gains of up to 50%, 3) BLT models unlock a new dimension for scaling LLMs, where model size can now be scaled while maintaining a fixed-inference budget, 4) We demonstrate the improved robustness of BLT models to input noise and their awareness of sub-word aspects of input data that token-based LLMs miss. Accompanying code for training and inference of BLT can be found at `https://github.com/facebookresearch/blt`.

2.2 Patching: From Individual Bytes to Groups of Bytes

Segmenting bytes into *patches* allows BLT to dynamically allocate compute based on context. Figure 2.3 shows several different methods for segmenting bytes into patches. Formally, a patching function f_p segments a sequence of bytes $\boldsymbol{x} = \{x_i, | i = 1, \ldots n\}$ of length n into a sequence of $m < n$ patches $\boldsymbol{p} = \{p_j | j = 1, \ldots, m\}$ by mapping each x_i to the set {0,1} where 1 indicates the start of a new patch. For both token-based and patch-based models, the computational cost of processing data is primarily determined by the number of steps executed by the main Transformer. In BLT, this is the number of patches

[1] We calculate the computational cost of a model by counting the number of Floating Point OPerations (FLOPs) needed.

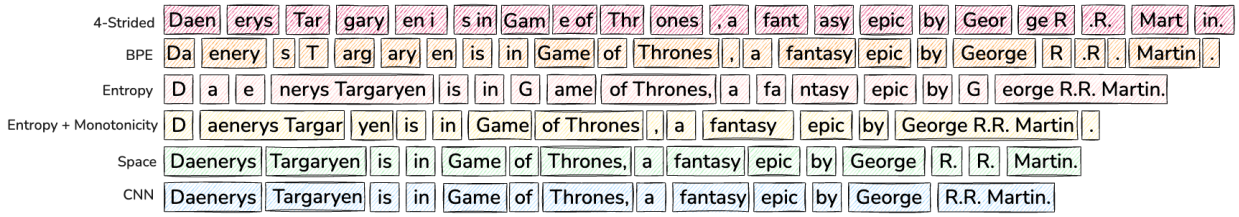

Figure 2.3: Patching schemes group bytes in different ways, each leading to a different number of resulting patches. Since each patch is processed using a large transformer step, the number of patches directly determines the bulk of the compute expended in terms of FLOPs. These schemes group bytes into patches by (a) striding every four bytes (§2.2.1) as in MegaByte [Yu et al., 2023], (b) tokenizing with Byte-Pair Encoding (BPE), in this case the Llama-3 [Dubey et al., 2024] tokenizer, (c & d) entropy-based patching as in this work (§2.2.3), (e) patching on space-bytes [Slagle, 2024], (f) and patching on entropy using a small CNN byte-level model with 2-byte context.

needed to encode the data with a given patching function. Consequently, the average size of a patch, or simply *patch size*, is the main factor for determining the cost of processing data during both training and inference with a given patching function (§2.4.5). Next, we introduce three patching functions: patching with a fixed number of bytes per patch (§2.2.1), whitespace patching (§2.2.2), and dynamically patching with entropies from a small byte LM (§2.2.3). Finally, we discuss incremental patching and how tokenization is different from patching (§2.2.4).

2.2.1 Strided Patching Every K Bytes

Perhaps the most straightforward way to group bytes is into patches of fixed size k as done in MegaByte [Yu et al., 2023]. The fixed stride is easy to implement for training and inference, provides a straightforward mechanism for changing the average patch size, and therefore makes it easy to control the FLOP cost. However, this patching function comes with significant downsides. First, compute is not dynamically allocated to where it is needed most: one could be either wasting a transformer step j if only predicting whitespace in code, or not allocating sufficient compute for bytes dense with information such as math. Second, this leads to inconsistent and non-contextual patching of similar byte sequences, such as the same word being split differently.

2.2.2 Space Patching

Slagle [2024] proposes a simple yet effective improvement over strided patching that creates new patches after any space-like bytes[2] which are natural boundaries for linguistic units in many languages. In Space patching, a latent transformer step (i.e., more FLOPs) is allocated to model every word. This ensures words are patched in the same way across sequences and that flops are allocated for hard predictions which often follow spaces. For example, predicting the first byte of the answer to the question "Who composed the Magic Flute?_" is much harder than predicting the remaining bytes after "M" since the first character significantly reduces the number of likely choices, making the completion "Mozart" comparatively easy to predict. However, space patching cannot gracefully handle all languages and domains, and most importantly cannot vary the patch size. Next, we introduce a new patching method that uses the insight that the first bytes in words are typically most difficult to predict, but that provides a natural mechanism for controlling patch size.

2.2.3 Entropy Patching: Using Next-Byte Entropies from a Small Byte LM

Rather than relying on a rule-based heuristic such as whitespace, we instead take a data-driven approach to identify high uncertainty next-byte predictions. We introduce *entropy patching*, which uses entropy estimates to derive patch boundaries.

We train a small byte-level auto-regressive language model on the training data for BLT and compute next byte entropies under the LM distribution p_e over the byte vocabulary $\mathcal{V}$:

$$H(x_i) = \sum_{v \in \mathcal{V}} p_e(x_i = v|\boldsymbol{x}_{<i}) \log p_e(x_i = v|\boldsymbol{x}_{<i}) \tag{2.1}$$

We experiment with two methods to identify patch boundaries given entropies $H(x_i)$. The first, finds points above a global entropy threshold, as illustrated in Figure 2.4. The second, identifies points that are high relative to the previous entropy. The second approach can also be interpreted as identifying points that

[2]Space-like bytes are defined as any byte that is not a latin character, digit, or UTF-8 continuation byte. In addition, each patch must contain at least one non space-like byte.

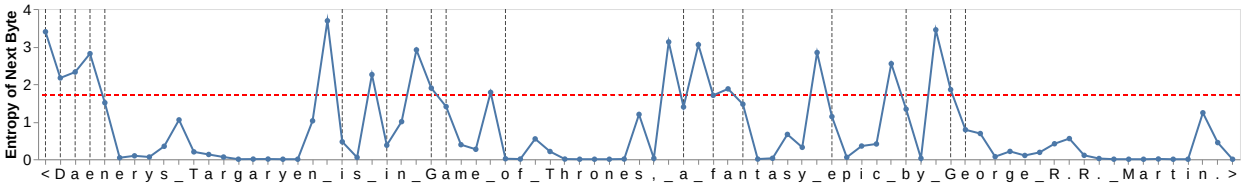

Figure 2.4: This figure plots the entropy $H(x_i)$ of each byte in "Daenerys Targeryen is in Game of Thrones, a fantasy epic by George R.R. Martin." with spaces shown as underscores. Patches end when $H(x_i)$ exceeds the global threshold θ_g, shown as a red horizontal line. The start of new patches are shown with vertical gray lines. For example, the entropies of "G" and "e" in "George R.R. Martin" exceed θ_g, so "G" is the start of a single byte patch and "e" of a larger patch extending to the end of the named entity as the entropy $H(x_i)$ stays low, resulting in no additional patches.

break approximate monotonically decreasing entropy withing the patch.

$$\text{Global Constraint} \qquad H(x_t) > \theta_g$$

$$\text{Approx. Monotonic Constraint} \qquad H(x_t) - H(x_{t-1}) > \theta_r$$

Patch boundaries are identified during a lightweight preprocessing step executed during dataloading. This is different from Nawrot et al. [2023] where classifier is trained to predict entropy-based patch boundaries. In our experiments (§2.4), we compare these two methods for distinguishing between low and high entropy bytes.

2.2.4 The Byte-Pair Encoding (BPE) Tokenizer and Incremental Patching

Many modern LLMs, including our baseline Llama 3, use a subword tokenizer like BPE [Gage, 1994; Sennrich et al., 2016]. We use "tokens" to refer to byte-groups drawn from a *finite* vocabulary determined prior to training as opposed to "patches" which refer to dynamically grouped sequences without a fixed vocabulary. A critical difference between patches and tokens is that with tokens, the model has no direct access to the underlying byte features.

A crucial improvement of BLT over tokenization-based models is that redefines the trade off between the vocabulary size and compute. In standard LLMs, increasing the size of the vocabulary means larger tokens on average and therefore fewer steps for the model but also larger output dimension for the final projection layer of the model. This trade off effectively leaves little room for tokenization based approaches to achieve

significant variations in token size and inference cost. For example, Llama 3 increases the average token size from 3.7 to 4.4 bytes at the cost of increasing the size of its embedding table 4x compared to Llama 2.

When generating, BLT needs to decide whether the current step in the byte sequence is at a patch boundary or not as this determines whether more compute is invoked via the Latent Transformer. This decision needs to occur independently of the rest of the sequence which has yet to be generated. Thus patching cannot assume access to future bytes in order to choose how to segment the byte sequence. Formally, a patching scheme f_p satisfies the property of incremental patching if it satisfies:

$$f_p(\boldsymbol{x}_{<i}) = f_p(\boldsymbol{x})_{<i}$$

BPE is not an incremental patching scheme as the same prefix can be tokenized differently depending on the continuation sequence, and therefore does not satisfy the property above[3].

2.3 BLT Architecture

BLT is composed of a large global autoregressive language model that operates on patch representations, along with two smaller local models that encode sequences of bytes into patches and decode patch representations back into bytes (Figure 2.2).

2.3.1 Latent Global Transformer Model

The Latent Global Transformer is an autoregressive transformer model $\mathcal{G}$ with $l_{\mathcal{G}}$ layers, which maps a sequence of latent input patch representations, p_j into a sequence of output patch representations, o_j. Throughout the chapter, we use the subscript j to denote patches and i to denote bytes. The global model uses a block-causal attention mask [Dubey et al., 2024], which restricts attention to be up to and including the current patch within the current document. This model consumes the bulk of the FLOPs during pre-training as well as inference, and thus, choosing when to invoke it allows us to control and vary the amount of compute expended for different portions of the input and output as a function of input/output complexity.

[3]Using a special delimiter token to indicate patch boundaries can turn BPE into an incremental patching scheme but increases the byte-sequence length.

2.3.2 Local Encoder

The Local Encoder Model, denoted by $\mathcal{E}$, is a lightweight transformer-based model with $l_{\mathcal{E}} << l_{\mathcal{G}}$ layers, whose main role is to efficiently map a sequence of input bytes b_i, into expressive patch representations, p_j. A primary departure from the transformer architecture is the addition of a cross-attention layer after each transformer layer, whose function is to pool byte representations into patch representations (Figure 2.5). First, the input sequence of bytes, b_i, are embedded using a $\mathbb{R}^{256\times h_{\mathcal{E}}}$ matrix, denoted as x_i. These embeddings are then optionally augmented with additional information in the form of hash-embeddings (§2.3.2). A series of alternating transformer and cross-attention layers (§2.3.2) then transform these representations into patch representations, p_i that are processed by the global transformer, $\mathcal{G}$. The transformer layers use a *local block causal* attention mask; each byte attends to a fixed window of $w_{\mathcal{E}}$ preceding bytes that in general can cross the dynamic patch boundaries but can not cross document boundaries. The following subsections describe details about the embeddings and the cross-attention block.

Encoder Hash n-gram Embeddings

A key component in creating robust, expressive representations at each step i is to incorporate information about the preceding bytes. In BLT, we achieve this by modeling both the byte b_i individually *and* as part of a byte n-gram. For each step i, we first construct byte-grams

$$g_{i,n} = \{b_{i-n+1}, \ldots, b_i\} \tag{2.2}$$

for each byte position i and n from three to eight.[4]

We then introduce hash n-gram embeddings, that map all byte n-grams via a hash function to an index in an embedding table E_n^{hash} with a fixed size, for each size $n \in \{3, 4, 5, 6, 7, 8\}$ [Bai et al., 2010]. The resulting embedding is then added to the embedding of the byte before being normalized and passed as input

[4]We omit byte-grams of size n or more when $i < n$.

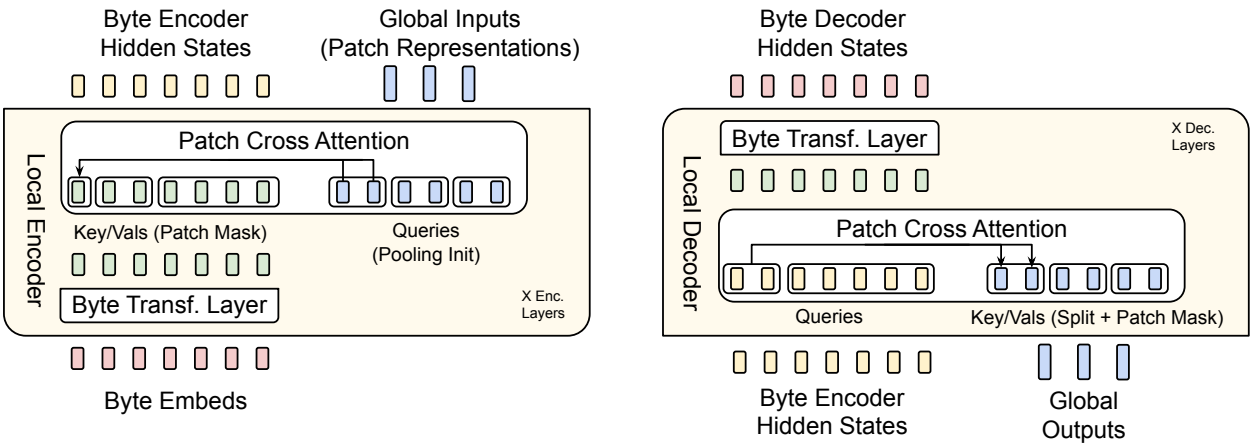

Figure 2.5: The local encoder uses a cross-attention block with patch representations as queries, and byte representations as keys/values to encode byte representations into patch representations. The local decoder uses a similar block but with the roles reversed i.e. byte representations are now the queries and patch representations are the keys/values. Here we use Cross-Attn $k = 2$.

to the local encoder model. We calculate the augmented embedding

$$e_i = x_i + \sum_{n=3,\ldots,8} E_n^{hash}(\text{Hash}(g_{i,n})) \tag{2.3}$$

$$\text{where, Hash}(g_{i,n}) = \text{RollPolyHash}(g_{i,n})\%|E_n^{hash}| \tag{2.4}$$

We normalize e_i by the number of n-grams sizes plus one and use RollPolyHash as defined in Appendix 6.3. In Section 2.7, we ablate the effects of n-gram hash embeddings with different values for n and embedding table size on FLOP-controlled scaling law trends. In addition to hash n-gram embeddings, we also experimented with frequency based n-gram embeddings, and we provide details of this exploration in Appendix 6.4.

Encoder Multi-Headed Cross-Attention

We closely follow the input cross-attention module of the Perceiver architecture [Jaegle et al., 2021], with the main difference being that latent representations correspond to variable patch representations as opposed to a fixed set of latent representations (Figure 2.5), and only attend to the bytes that make up the respective patch. The module comprises a query vector, corresponding to each patch p_j, which is initialized by pooling

the byte representations corresponding to patch p_j, followed by a linear projection, $\mathcal{E}_C \in \mathbb{R}^{h_\mathcal{E} \times (h_\mathcal{E} \times U_\mathcal{E})}$, where $U_\mathcal{E}$ is the number of encoder cross-attention heads. Formally, if we let $f_{\text{bytes}}(p_j)$ denote the sequence of bytes corresponding to patch, p_j, then we calculate

$$P_{0,j} = \mathcal{E}_C(f_{\text{bytes}}((p_j)), f \text{ is a pooling function} \tag{2.5}$$

$$P_l = P_{l-1} + W_o \left(\text{softmax} \left(\frac{QK^T}{\sqrt{d_k}} \right) V \right) \tag{2.6}$$

$$\text{where } Q_j = W_q(P_{l-1,j}), K_i = W_k(h_{l-1,i}), V_i = W_v(h_{l-1,i}) \tag{2.7}$$

$$h_l = \text{Encoder-Transformer-Layer}_l(h_{l-1}) \tag{2.8}$$

where $P \in \mathbb{R}^{n_p \times h_\mathcal{G}}$ represents n_p patch representations to be processed by the global model, which is initialized by pooling together the byte embeddings e_i corresponding to each patch p_j. W_q, W_k, W_v and W_o are the projections corresponding to the queries, keys, values, and output where the keys and values are projections of byte representations h_i from the previous layer (e_i for the first layer). We use a masking strategy specific to patching where each query Q_j only attends to the keys and values that correspond to the bytes in patch j. Because we use multi-headed attention over Q, K and V and patch representations are typically of larger dimension ($h_\mathcal{G}$) than $h_\mathcal{E}$, we maintain P_l as multiple heads of dimension $h_\mathcal{E}$ when doing cross-attention, and later, concat these representations into $h_\mathcal{G}$ dimensions. Additionally, we use a pre-LayerNorm on the queries, keys and values and no positional embeddings are used in this cross-attention module. Finally, we use a residual connection around the cross-attention block.

2.3.3 Local Decoder

Similar to the local encoder, the local decoder $\mathcal{D}$ is a lightweight transformer-based model with $l_\mathcal{D} << l_\mathcal{G}$ layers, that decodes a sequence of global patch representations o_j, into raw bytes, y_i. The local decoder predicts a sequence of raw bytes, as a function of previously decoded bytes, and thus, takes as input the hidden representations produced by the local encoder for the byte-sequence. It applies a series of $l_\mathcal{D}$ alternating layers of cross attention and transformer layers. The cross-attention layer in the decoder is applied before the transformer layer to first create byte representations from the patch representations, and the local decoder transformer layer operates on the resulting byte sequence.

Decoder Multi-headed Cross-Attention

In the decoder cross-attention, the roles of the queries and key/values are interchanged i.e. the byte-representations are now the queries, and the patch representations are now the key/values. The initial byte-representations for the cross-attention are initialized as the byte embeddings from the last encoder layer i.e. $h_{l_\mathcal{E}}$. The subsequent byte-representations for layer l, $d_{l,i}$ are computed as:

$$D_0 = h_{l_\mathcal{E}} \tag{2.9}$$

$$B_l = D_{l-1} + W_o \left(\text{softmax} \left(\frac{QK^T}{\sqrt{d_k}} \right) V \right), \tag{2.10}$$

$$\text{where } Q_i = W_q(d_{l-1,i}), K_i = W_k(\mathcal{D}_C(o_j)), V_i = W_v(\mathcal{D}_C(o_j)) \tag{2.11}$$

$$D_l = \text{Decoder-Transformer-layer}_l(B_l) \tag{2.12}$$

where once again, W_k, W_v are key/value projection matrices that operate on a linear transformation and split operation $\mathcal{D}_C$, applied to the final patch representations o_j from the global model, W_q is a query projection matrices operating on byte representations d_{l-1} from the previous decoder transformer layer (or $h_{l_\mathcal{E}}$ for the first layer), and W_o is the output projection matrix, thus making $B \in \mathbb{R}^{h_\mathcal{D} \times n_b}$, where n_b is the number of output bytes. The next decoder representations D_l are computed using a decoder transformer layer on the output of the cross-attention block, B. As in the local encoder cross-attention, we use multiple heads in the attention, use pre LayerNorms, no positional embeddings, and a residual connection around the cross-attention module.

2.4 Experimental Setup

We carefully design controlled experiments to compare BLT with tokenization based models with particular attention to not give BLT any advantages from possibly using longer sequence contexts.

2.4.1 Pre-training Datasets

All model scales that we experiment in this chapter are pre-trained on two datasets: 1) The Llama 2 dataset [Touvron et al., 2023], which comprises 2 trillion tokens collected from a variety of publicly available sources, which are subsequently cleaned and filtered to improve quality; and 2) BLT-1T: A new dataset with 1 trillion tokens gathered from various public sources, and also including a subset of the pre-training data released by Datacomp-LM [Li et al., 2024]. The former is used for scaling law experiments on optimal number of tokens as determined by Dubey et al. [2024] to determine the best architectural choices for BLT, while the latter is used for a complete pre-training run to compare with Llama 3 on downstream tasks. Neither of these datasets include any data gathered from Meta products or services. Furthermore, for baseline experiments for tokenizer-based models, we use the Llama 3 tokenizer with a vocabulary size of 128K tokens, which produced stronger baseline performance that the Llama 2 tokenizer in our experiments.

2.4.2 Entropy Model

The entropy model in our experiments is a byte level language model trained on the same training distribution as the full BLT model. Unless otherwise mentioned, we use a transformer with 100M parameters, 14 layers, and a hidden dimensionality of 512, and sliding window attention of 512 bytes. The remaining hyperparameters are the same as in our local and global transformers. We experimented with different model sizes, receptive fields, and architectures as discussed in section 2.7. In particular, when the receptive field of the model is small enough, the trained entropy model can be encoded in an efficient lookup table.

2.4.3 Entropy Threshold and Equalizing Context Length

For models using entropy-based patching, we estimate a patching threshold that achieves a desired average *patch size* on the pretraining data mix. In BLT, unlike with tokenization, the *patch size* can be arbitrarily chosen having significant implications on the context size used by the model. To maintain the same average context length and avoid giving larger patch sizes unfair advantage, we ensure that the number of bytes in each batch remains constant in expectation. This means that we reduce the sequence length of models with larger patch sizes. On Llama 2 data, we use a 8k byte context while on the BLT-1T dataset we increase the context to 16k bytes on average while maintaining the same batch size of 16M bytes on average.

While the average batch size is constant, when loading batches of data, dynamic patching methods yield different ratios of bytes to patches. For efficiency reasons, our implementation of BLT training packs batches of patches to avoid padding steps in the more expensive latent transformer. This ensures that every batch has the same number of patches. During training we pad and possibly truncate byte sequences to 12k and 24k bytes respectively for Llama 2 and BLT-1T datasets, to avoid memory spikes from sequences with unusually large patches.

2.4.4 Entropy Model Context

Empirically, we find that using entropy patching yields progressively larger patches in structured content like multiple choice tasks (see patching on an MMLU example in Figure 6.1) which are often very repetitive. These variations are caused by lower entropy on the repeated content found in the entropy model context. So for the large scale run of BLT-Entropy with patch size 4.5, we reset the entropy context with new lines and use approximate monontonicity constraint as it suffers less from "entropy drift" from changes in context length. This change only affects how we compute entropies, but we still follow the same procedure to identify the value of the entropy threshold.

2.4.5 FLOPs Estimation

We largely follow the equations for computation of transformer FLOPs from Chinchilla [Hoffmann et al., 2022] comprising FLOPs for the feed-forward layers, QKVO projections in the self-attention layer, and computation of attention and output projection. A notable difference is that we assume the input embedding layer is implemented as an efficient lookup instead of a dense matrix multiplication, therefore becoming a 0-FLOP operation. Following previous work, we estimate that the backwards pass has twice the number of FLOPs as the forward pass.

To compute FLOPs *per byte* for BLT models, we add up the FLOPs for the local encoder transformer, the global latent transformer, and the local decoder transformer, together with the cross attention blocks in the encoder and the decoder:

$$\text{FL}_{\text{BLT}} = \text{Transf. FL}(h_{\mathcal{G}}, l_{\mathcal{G}}, m = n_{ctx}/n_p, V = 0)/n_p \tag{2.13}$$

$$+ \text{Transf. FL}(h_{\mathcal{E}}, l_{\mathcal{E}}, m = w_{\mathcal{E}}, V = 0) \tag{2.14}$$

$$+ \text{Transf. FL}(h_{\mathcal{D}}, l_{\mathcal{D}}, m = w_{\mathcal{D}}, V = 256) \tag{2.15}$$

$$+ \text{Cross Attn. FL}(h_{\mathcal{E}}, l_{\mathcal{E}}, m = n_p, r = n_p/k) \times k/n_p \tag{2.16}$$

$$+ \text{Cross Attn. FL}(h_{\mathcal{D}}, l_{\mathcal{D}}, m = k, r = k/n_p) \tag{2.17}$$

where n_{ctx} is the sequence length in bytes, n_p is the patch size, r is the ratio of queries to key/values, k is the ratio of patch-dimension to byte-dimension i.e. the number of local model splits that concatenate to form a global model representation ($k = 2$ in Figure 2.5). V corresponds to the vocabulary size for the output projection, which is only used in the local decoder. Depending on whether a module is applied on the byte or patch sequence, the attention uses a different context length, m. We modify the attention FLOPs accordingly for each component. The exact equations for FLOPs computation for Transformer-FLOPs and Cross-Attention FLOPs are provided in Appendix 6.2.

2.4.6 Bits-Per-Byte Estimation

Perplexity only makes sense in the context of a fixed tokenizer as it is a measure of the uncertainty for each token. When comparing byte and token-level models, following previous work [Xue et al., 2022; Yu et al., 2023; Wang et al., 2024], we instead report Bits-Per-Byte (BPB), a tokenizer independent version of perplexity. Specifically:

$$\text{BPB}(x) = \frac{\mathcal{L}_{CE}(\boldsymbol{x})}{\ln(2) \cdot n_{\text{bytes}}} \tag{2.18}$$

where the uncertainty over the data $\boldsymbol{x}$ as measured by the sum of the cross-entropy loss is normalized by the total number of bytes in $\boldsymbol{x}$ and a constant.

2.4.7 Transformer Architecture Hyperparameters

For all the transformer blocks in BLT, i.e. both local and global models, we largely follow the architecture of Llama 3 [Dubey et al., 2024]; we use the SwiGLU activation function [Shazeer, 2020] in the feed-forward layers, rotary positional embeddings (RoPE) [Su et al., 2024] with $\theta = 500000$ [Xiong et al., 2024] only in self-attention layers, and RMSNorm [Zhang and Sennrich, 2019] for layer normalization. We use Flash attention [Dao et al., 2022] for all self-attention layers that use fixed-standard attention masks such as *block causal* or *fixed-window block causal*, and a window size of 512 for fixed-width attention masks. Since our cross-attention layers involve dynamic patch-dependent masks, we use Flex Attention[5] to produce fused implementations and significantly speed up training.

2.4.8 BLT-Specific Hyperparameters

To study the effectiveness of BLT models, we conduct experiments along two directions, scaling trends, and downstream task evaluations, and we consider models at different scales: 400M, 1B, 2B, 4B and 8B for these experiments. The architecture hyperparameters for these models are presented in Appendix Table 6.1. We use max-pooling to initialize the queries for the first cross-attention layer in the local encoder. We use $500,000$ hashes with a single hash function, with n-gram sizes ranging from 3 to 8, for all BLT models. We use a learning rate of $4e-4$ for all models. The choice of matching learning rate between token and BLT models follows a hyperparameter search between $1e-3$ and $1e-4$ at 400M and 1B model scales showing the same learning rate is optimal. For scaling trends on Llama-2 data, we use training batch-sizes as recommended by Dubey et al. [2024] or its equivalent in bytes. For optimization, we use the AdamW optimizer [Loshchilov and Hutter, 2019] with β_1 set to 0.9 and β_2 to 0.95, with an $\epsilon = 10^{-8}$. We use a linear warm-up of 2000 steps with an cosine decay schedule of the learning rate to 0, we apply a weight decay of 0.1, and global gradient clipping at a threshold of 1.0.

[5] `https://pytorch.org/blog/flexattention`

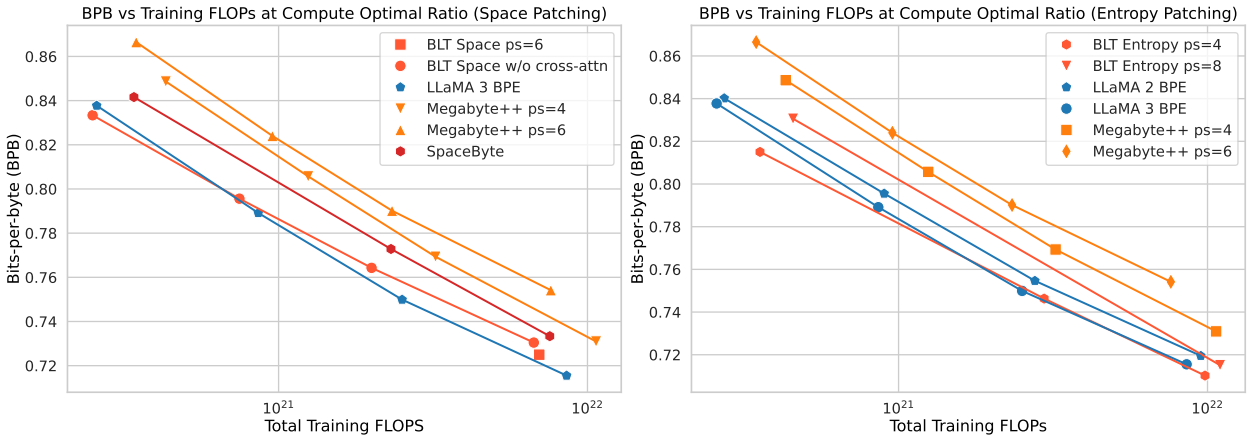

Figure 2.6: Scaling trends for BLT models with different architectural choices, as well as for baseline BPE token-based models. We train models at multiple scales from 1B up to 8B parameters for the optimal number of tokens as computed by Dubey et al. [2024] and report bits-per-byte on a sample from the training distribution. BLT models perform on par with state-of-the-art tokenizer-based models such as Llama 3, at scale. PS denotes patch size. We illustrate separate architecture improvements on space-patching (**left**) and combine them with dynamic patching (**right**).

2.5 Scaling Trends

We present a holistic picture of the scaling trends of byte-level models that can inform further scaling of BLT models. Our scaling study aims to address the limitations of previous research on byte-level models in the following ways: (a) We compare trends for the compute-optimal training regime, (b) We train matching 8B models on non-trivial amounts of training data (up to 1T tokens/4T bytes) and evaluate on downstream tasks, and (c) We measure scaling trends in inference-cost controlled settings. In a later section, we will investigate specific advantages from modeling byte-sequences.

2.5.1 Parameter Matched Compute Optimal Scaling Trends

Using the Llama 2 dataset, we train various *compute-optimal* BPE and BLT models across four different sizes, ranging from 1B to 8B parameters. We then plot the training FLOPs against language modeling performance on a representative subset of the training data mixture. The BPE models are trained using the optimal ratio of model parameters to training data, as determined by Llama 3 [Dubey et al., 2024]. This *compute-optimal* setup is theoretically designed to achieve the best performance on the training dataset within a given training budget [Hoffmann et al., 2022], providing a robust baseline for our model. For each

BPE model, we also train a corresponding BLT model on the same data, using a Latent Transformer that matches the size and architecture of the corresponding BPE Transformer.

As illustrated in Figure 2.6 (right), BLT models either match or outperform their BPE counterparts and this trend holds as we scale model size and FLOPs. To the best of our knowledge, BLT is the first byte-level Transformer architecture to achieve matching scaling trends with BPE-based models at compute optimal regimes. This therefore validates our assumption that the optimal ratio of parameters to training compute for BPE also applies to BLT, or at least it is not too far off.

Both architectural improvements and dynamic patching are crucial to match BPE scaling trends. In Figure 2.6 (left), we compare space-patching-based models against Llama 3. We approximate SpaceByte [Slagle, 2024] using BLT space-patching without n-gram embeddings and cross-attention. Although SpaceByte improves over Megabyte, it remains far from Llama 3. In Figure 2.6 (right), we illustrate the improvements from both architectural changes and dynamic patching. BLT models perform on par with state-of-the-art tokenizer-based models such as Llama 3, at scale.

We also observe the effects of the choice of tokenizer on performance for tokenizer-based models, i.e., models trained with the Llama-3 tokenizer outperform those trained using the Llama-2 tokenizer on the same training data.

Finally, our BLT architecture trends between Llama 2 and 3 when using significantly larger patch sizes. The BPE tokenizers of Llama 2 and 3 have an average token size of 3.7 and 4.4 bytes. In contrast, BLT can achieve similar scaling trends with an average patch size of 6 and even 8 bytes. Inference FLOP are inversely proportional to the average patch size, so using a patch size of 8 bytes would lead to nearly 50% inference FLOP savings. Models with larger patch sizes also seem to perform better as we scale model and data size. BLT with patch size of 8 starts at a significantly worse point compared to BPE Llama 2 at 1B but ends up better than BPE at 7B scale. This suggests that such patch sizes might perform better at even larger scales and possibly that even larger ones could be feasible as model size and training compute grow.

2.5.2 Beyond Compute Optimal Task Evaluations

To assess scaling properties further, we train an 8B BLT model beyond the compute optimal ratio on the BLT-1T dataset, a larger higher-quality dataset, and measure performance on a suite of standard classifica-

	Llama 3 1T Toks	BLT-Space 6T Bytes	BLT-Global 4.5T Bytes	BLT-Mono 4.5T Bytes
Arc-E	77.6	75.4	76.4	**79.6**
Arc-C	**53.3**	49.8	52.1	52.1
HellaSwag	79.1	79.6	80.4	**80.6**
PIQA	80.7	81.1	**81.3**	80.6
MMLU	**58.1**	54.8	56.2	57.4
MBPP	40.2	37.6	**42.2**	41.8
HumanEval	31.1	27.4	29.3	**35.4**
Average	60.0	58.0	59.7	**61.1**
Bytes/Patch Train Mix	4.4	**6.1**	4.5	4.5

Table 2.1: Comparison of FLOP-matched BLT and BPE 8B models trained on the BLT-1T dataset on downstream tasks. BLT outperforms Llama 3, and depending on the patching scheme, achieves significant FLOPs savings at the expense of minor performance reduction.

tion and generation benchmarks. For evaluation, we select the following common sense reasoning, world knowledge, and code generation tasks:

Classification tasks include ARC-Easy (0-shot) [Clark et al., 2018], Arc-Challenge (0-shot) [Clark et al., 2018], HellaSwag (0-shot) [Zellers et al., 2019], PIQA (0-shot) [Bisk et al., 2020], and MMLU (5-shot) [Hendrycks et al., 2021]. We employ a prompt-scoring method, calculating the likelihood over choice characters, and report the average accuracy.

Coding related generation tasks: We report pass@1 scores on MBPP (3-shot) [Austin et al., 2021] and HumanEval (0-shot) [Chen et al., 2021], to evaluate the ability of LLMs to generate Python code.

In Table 2.1, we compare three models trained on the BLT-1T dataset: a BPE Llama 3 tokenizer-based model,[6] and two variants of the BLT model. One employing a space-patching scheme (BLT-Space) and another utilizing an entropy-based patching scheme (BLT-Entropy). with approx. monotonicity constraint and reset the context of the entropy model with new lines (as discussed in subsection 2.4.4). All three models are trained with an equivalent FLOP budget. However, with BLT-Entropy we additionally make an inference time adjustment of the entropy threshold from 0.6 to 0.1 which we find to improve task performance at the cost of more inference steps.

The BLT-Entropy model outperforms the Llama 3 model on 4 out of 7 tasks while being trained on the same number of bytes. This improvement is like due to a combination of (1) a better use of training compute via dynamic patching, and (2) the direct modeling of byte-level information as opposed to tokens.

[6]We choose the Llama 3 tokenizer with its 128k vocabulary as it performs better than Llama 2's 32k vocabulary.

Llama 2	Llama 3	Entropy ps=6	Entropy ps=8	Inference FLOPs	Compute Optimal (Bytes)	Crossover (Bytes)
470m	450m	610m (1.2x)	760m (1.6x)	3.1E8	50B	150B
3.6B	3.9B	5.2B (1.3x)	6.6B (1.7x)	2.1E9	400B	1T

Table 2.2: Details of models used in the fixed-inference scaling study. We report non-embedding parameters for each model and their relative number compared to Llama 2. We pick model sizes with equal inference FLOPs per byte. We also indicate BPE's compute-optimal training data quantity and the crossover point where BLT surpasses BPE as seen in Figure 2.1 (both expressed in bytes of training data). This point is achieved at much smaller scales compared to many modern training budgets.

On the other hand, BLT-Space underperforms the Llama 3 tokenizer on all but one task, but it achieves a significant reduction in inference FLOPs with its larger average patch size of 6 bytes. In comparison, the BPE and entropy-patching based models have roughly equivalent average patch size of approximately 4.5 bytes on the training data mix. With the same training budget, the larger patch size model covers 30% more data than the other two models which might push BLT further away from the compute-optimal point.

2.5.3 Patches Scale Better Than Tokens

With BLT models, we can simultaneously increase model size and patch size while maintaining the same training and inference FLOP budget and keeping the amount of training data constant. Arbitrarily increasing the patch size is a unique feature of patch-based models which break free of the efficiency tradeoffs of fixed-vocabulary token-based models, as discussed in Section 2.2.4. Longer patch sizes save compute, which can be reallocated to grow the size of the global latent transformer, because it is run less often.

We conduct a fixed inference scaling study to test the hypothesis that larger models taking fewer steps on larger patches might perform better than smaller models taking more steps. Starting from model sizes of 400m and 3.6B parameters with the Llama 2 tokenizer, we find FLOP equivalent models with the Llama 3 tokenizer and BLT-Entropy models with average patch sizes of 6 and 8 bytes on the training datamix (see Table 2.2 for model details). For patch size 8 models, we use 3 encoder layers instead of 1. We train each model for various training FLOP budgets.

Figure 2.1 shows that BLT models achieve better scaling trends than tokenization-based architectures for both inference FLOP classes. In both cases, BPE models perform better with small training budgets and are quickly surpassed by BLT, not far beyond the compute-optimal regime. In practice, it can be preferable to spend more during the one-time pretraining to achieve a better performing model with a fixed inference

budget. A perfect example of this is the class of 8B models, like Llama 3.1, which has been trained on two orders of magnitude more data than what is compute-optimal for that model size.

The crossover point where BLT improves over token-based models has shifted slightly closer to the compute-optimal point when moving to the larger FLOP class models (from 3x down to 2.5x the compute optimal budget). Similarly, the larger patch size 8 model has steeper scaling trend in the larger FLOP class overtaking the other models sooner. As discussed in Section 2.5.1, larger patch sizes appear to perform closer to BPE models at larger model scales. We attribute this, in part, to the decreasing share of total FLOPs used by the byte-level Encoder and Decoder modules which seem to scale slower than the Latent Transformer. When growing total parameters 20x from 400M to 8B, we only roughly double BLT's local model parameters. This is important as larger patch sizes only affect FLOPs from the patch Latent Transformer and not the byte-level modules. In fact, that is why the BLT-Entropy ps=8 went from 1.6x to 1.7x of the Llama 2 model size when moving to the larger model scale.

In summary, our patch-length scaling study demonstrates that the BLT patch-based architecture can achieve better scaling trends by simultaneously increasing both patch and model size. Such trends seem to persist and even improve at larger model scales.

2.6 Byte Modeling Improves Robustness

We also measure the robustness of BLT compared to token-based models that lack direct byte-level information, and present an approach to byte-ify pretrained token-based models.

2.6.1 Character-Level Tasks

A very early motivation for training byte-level models was to take advantage of their robustness to byte level noise in the input, and also to exploit their awareness of the constituents of tokens, which current tokenizer-based models struggle with. To measure these phenomena, we perform additional evaluations on benchmarks that evaluate both robustness to input noise as well as awareness of characters, both English and multi-lingual, including digits and phonemes. We present these results in Table 2.3.

Noisy Data We create noised versions of the benchmark classification tasks described in Section 2.5.2, to compare the robustness of tokenizer-based models with that of BLT. We employ five distinct character-level noising strategies to introduce variations in the text: (a) *AntSpeak*: This strategy converts the entire text into uppercase, space-separated characters. (b) *Drop*: Randomly removes 10% of the characters from the text. (c) *RandomCase*: Converts 50% of the characters to uppercase and 50% to lowercase randomly throughout the text. (d) *Repeat*: Repeats 20% of the characters up to a maximum of four times. (e) *UpperCase*: Transforms all characters in the text to uppercase. During evaluation, we apply each noising strategy to either the prompt, completion, or both as separate tasks and report the average scores. In Table 2.3 we report results on noised HellaSwag [Zellers et al., 2019] and find that BLT indeed outperforms tokenizer-based models across the board in terms of robustness, with an average advantage of 8 points over the model trained on the same data, and even improves over the Llama 3.1 model trained on a much larger dataset.

Phonology - Grapheme-to-Phoneme (G2P) We assess BLT's capability to map a sequence of graphemes (characters representing a word) into a transcription of that word's pronunciation (phonemes). In Table 2.3, we present the results of the G2P task in a 5-shot setting using Phonology Bench [**?**] and find that BLT outperforms the baseline Llama 3 1T tokenizer-based model on this task.

CUTE To assess character-level understanding, we evaluate BLT on the CUTE benchmark [Edman et al., 2024], which comprises several tasks that are broadly classified into three categories: understanding composition, understanding orthographic similarity, and ability to manipulate sequences. This benchmark poses a significant challenge for most tokenizer-based models, as they appear to possess knowledge of their tokens' spellings but struggle to effectively utilize this information to manipulate text. Table 2.3 shows that BLT-Entropy outperforms both BPE Llama 3 models by more than 25 points on this benchmark. In particular, our model demonstrates exceptional proficiency in character manipulation tasks achieving 99.9% on both spelling tasks. Such large improvements despite BLT having been trained on 16x less data than Llama 3.1 indicates that character level information is hard to learn for BPE models. Figure 2.7 illustrates a few such scenarios where Llama 3 tokenizer model struggles but our BLT model performs well. Word deletion and insertion are the only two tasks where BPE performs better. Such word manipulation might not be straightforward for a byte-level model but the gap is not too wide and building from characters to words could be

Task	Prompt	Llama 3	BLT
Substitute Word	Question: Substitute " and " with " internet " in " She went to the kitchen and saw two cereals. ". Answer:	She went to the kitchen and saw two cereals.	She went to the kitchen internet saw two cereals.
Swap Char	Question: Swap " h " and " a " in " that ". Answer:	that	taht
Substitute Char	Question: Substitute " a " with " m " in " page ". Answer:	-	pmge
Semantic Similarity	Question: More semantically related to " are ": " seem ", " acre ". Answer:	acre	seem
Orthographic Similarity	Question: Closer in Levenshtein distance to " time ": " timber ", " period ". Answer:	period	timber
Insert Char	Question: Add an " z " after every " n " in " not ". Answer:	znotz	nzot

Figure 2.7: Output responses from Llama 3 and BLT models for various tasks from CUTE benchmark. BLT model performs better on sequence manipulation tasks compared to the tokenizer-based Llama 3 model. Note that few-shot examples are not shown in the above prompts to maintain clarity.

easier than the other way around. We use the same evaluation setup in all tasks and the original prompts from Huggingface. BPE models might benefit from additional prompt engineering.

Low Resource Machine Translation We evaluate BLT on translating into and out of six popular language families and twenty one lower resource languages with various scripts from the FLORES-101 benchmark [Goyal et al., 2022a] and report SentencePiece BLEU in Table 2.4. Our results demonstrate that BLT outperforms a model trained with the Llama 3 tokenizer, achieving a 2-point overall advantage in translating into English and a 0.5-point advantage in translating from English. In popular language pairs, BLT performs comparably to or slightly better than Llama 3. However, BLT outperforms Llama 3 on numerous language pairs within lower-resource language families, underscoring the effectiveness of byte modeling for generalizing to long-tail byte sequences.

2.6.2 Training BLT from Llama 3

We explore a workflow where BLT models can leverage existing pre-trained tokenizer-based models for better and faster training convergence, acheived by initializing the global transformer parameters of BLT with those of a pre-trained Llama 3.1 model. Subsequently, we update the weights of the global transformer using one-tenth the learning rate employed for the local encoder and local decoder model, for Llama 3 optimal number of steps, and present a comparison with a baseline BLT in Table 2.5. It is evident that BLT from Llama 3.1 significantly outperforms both the Llama 3 and BLT baselines, which were trained with the same number of FLOPs. Moreover, when compared to our BLT-Entropy model (as presented in Table 2.1), which was trained on a significantly larger dataset (1T tokens), BLT from Llama 3.1 still achieves superior performance on MMLU task, suggesting that it can be an effective approach in significantly reducing the training FLOPs.

This setup can also be viewed as transforming tokenizer-based models into tokenizer-free ones, effectively converting a pre-trained LLaMA 3.1 model into a BLT model. To provide a comprehensive comparison, we include the original LLaMA 3.1 model trained on 15T tokens in Table 2.5 and evaluate it against the BLT derived from LLaMA 3. Our model experiences a slight performance decline on MMLU and HumanEval, but a more significant drop on other tasks. This suggests that further work is needed to fully leverage the pre-trained model and improve upon its performance, particularly in terms of optimizing data

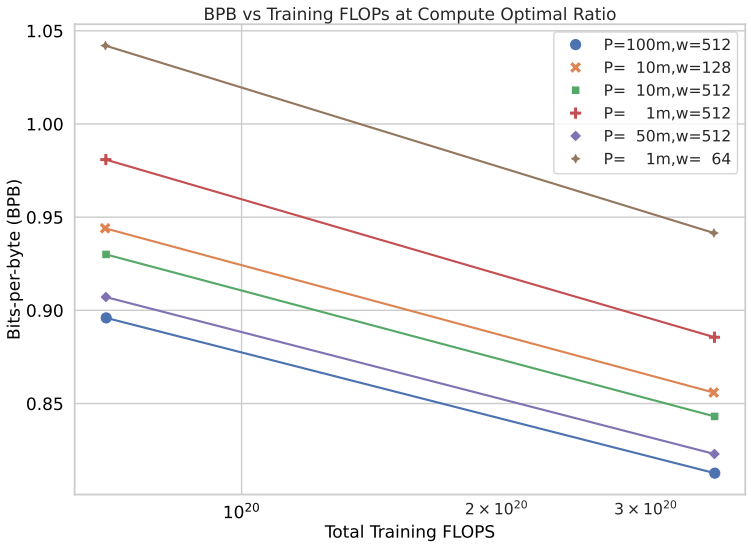

Figure 2.8: Variation of language modeling performance in bits-per-byte (bpb) with training FLOPs for 400m and 1b BLT models patched with entropy models of different sizes and context windows. Both dimensions improve scaling performance, with diminishing returns beyond 50m parameter entropy models with a context of 512 bytes.

mixtures and other hyperparameters.

2.7 Ablations and Discussion

In this section, we discuss ablations justifying architectural choices for BLT and the patching scheme and hyper-parameters for the BLT 8B parameter model trained on the BLT-1T dataset.

Entropy Model Hyper-parameters To study the effect of varying entropy model size and context window length on scaling performance, we train byte-level entropy transformer models of different model sizes between 1m and 100m parameters, with varying context window lengths from 64 to 512. We plot bpb vs training FLOP scaling law curves, created using our $400m$ and $1b$ BLT models trained on the Llama-2 dataset and present them in Figure 2.8. We find that scaling performance is positively correlated with both these dimensions of the entropy model, with diminishing returns when we scale beyond 50m parameters.

Types of Patching We ablate the four different patching schemes, introduced in Section 2.2 i.e. 1) Strided Patching with a stride of 4 and 6, 2) Patching on whitepsace, 3) BPE Tokenizer patching based on the Llama

3 tokenizer, and 4) Entropy based patching using a small byte LLM.

While dynamic patching reduces the effective length of sequences, we control for the sequence length to maintain a similar context length for all patching schemes. All the models see the same number of bytes in each sequence during training and inference in expectation to prevent any confounding factors from being able to model larger contexts. Figure 2.6 highlights the results of these ablations. All the remaining patching schemes outperform static patching, with space patching being a very close competitor to dynamic entropy based patching.

In Table 2.6, we present benchmark evaluations for BLT models comparing tokenizer-based models, space patching, and entropy-based patching, trained on the Llama 2 dataset for an optimal number of steps [Dubey et al., 2024]. Although space patching is a simpler strategy that does not involve running an entropy model on the fly during training, we find that the gains we observed using entropy-based patching on scaling trends (Section 2.5) do indeed carry forward even to downstream benchmark tasks.[7]

Cross-Attention In Table 2.7, we ablate including cross-attention at various points in the encoder and decoder of BLT. For the encoder cross-attention we test initializing the queries with 1) the same learned embedding for every global state, 2) a hash embedding of the bytes in the patch, and 3) pooling of the encoder hidden representation of the patch bytes at the given encoder layer.

We find that using cross-attention in the *decoder* is most effective. In the encoder, there is a slight improvement in using cross-attention but only with pooling initialization of queries. Additionally, we find that cross-attention helps particularly on Common-Crawl and especially with larger patch sizes.

n-gram Hash Embeddings We ablate settings of 0, 100K, 200K and 400K n-gram hash embedding vocabularies and present results in Table 2.8. We find that hash embeddings help on all domains, but particularly on Wikipedia and Github (0.04 bpb difference compared to 0.01 bpb difference after 15k steps at 8B). At 8B scale going from 500K to 300K hashes changed performance by 0.001 bpb on 15k steps. This indicates that hashes are vital to bringing the performance of BLT to match those of tokenizer based models, however, after 300K hashes, there are diminishing returns. Additionally, it appears that the gains are largely complementary with cross-attention as they provide improvements on different datasets.

[7] Space patching results are from earlier runs without cross-attention, but similar trends are observed even with cross-attention.

Local Model Hyperparamaters In Table 2.9, we ablate various settings for the number of layers in the local encoder and decoder. When paired with hash n-gram embeddings, BLT works well with an encoder that is extremely light-weight i.e. just one layer, and with a heavier decoder.

2.8 Related Work

Character-Level RNNs: Character Language Modeling has been a popular task ever since the early days of neural models [Sutskever et al., 2011; Mikolov et al., 2012; Graves, 2013] owing to their flexibility of modeling out of vocabulary words organically without resorting to back-off methods. Kim et al. [2016] also train a model that processes characters only on the input side using convolutional and highway networks that feed into LSTM-based RNNs and are able to match performance with the RNN based state-of-the-art language models of the time on English and outperform them on morphologically rich languages, another sought-after advantage of character-level LLMs. Kenter et al. [2018] do machine comprehension using byte-level LSTM models that outperformed word-level models again on morphologically-rich Turkish and Russian languages. Along similar lines, Al-Rfou et al. [2019] used character-based convolutional models for classification tasks, which outperformed word-level models for certain tasks. Chung et al. [2017] use hierarchical LSTM models using boundary-detectors at each level to discover the latent hierarchy in text, to further improve performance on character level language modeling. ByteNet by Kalchbrenner et al. [2016] uses CNN based layers on characters as opposed to attention for machine translation.

Character-Level Transformers: The development of transformer models using attention [Vaswani et al., 2017] together with subword tokenization [Sennrich et al., 2016], significantly improved the performance of neural models on language modeling and benchmark tasks. However, word and sub-word units implicitly define an inductive bias for the level of abstraction models should operate on. To combine the successes of transformer models with the initial promising results on character language modeling, Al-Rfou et al. [2019] use very deep transformers, and with the help of auxiliary losses, train transformer-based models that outperformed previous LSTM based character LLMs. However, they still saw a significant gap from word level LLMs. GPT-2 [Radford et al., 2019] also observed that on large scale datasets like the 1 billion word benchmark, byte-level LMs were not competitive with word-level LMs.

While Choe et al. [2019] demonstrated that byte-level LLMs based on transformers can outperform sub-

word level LLMs with comparable parameters, the models take up much more compute and take much longer to train. Similarly, El Boukkouri et al. [2020] train a BERT model (CharFormer) that builds word representations by applying convolutions on character embeddings, and demonstrate improvements on the medical domain, but they also expend much more compute in doing so. Clark et al. [2022] develop CANINE, a 150M parameter encoder-only model that operates directly on character sequences. CANINE uses a deep transformer stack at its core similar in spirit to our global model, and a combination of a local transformer and strided convolutions to downsample the input characters, and outperforms the equivalent token-level encoder-only model (mBERT) on downstream multilingual tasks. ByT5 [Xue et al., 2022] explored approaches for byte-level encoder decoder models, that do not use any kind of patching operations. While their model exhibited improved robustness to noise, and was competitive with tokenizer-based models with 4x less data, the lack of patching meant that the models needed to compute expensive attention operations over every byte, which was extremely compute heavy. Directly modeling bytes instead of subword units increases the sequence length of the input making it challenging to efficiently scale byte level models. Recently, using the Mamba Architecture [Gu and Dao, 2024], which can maintain a fixed-size memory state over a very large context length, Wang et al. [2024] train a byte-level Mamba architecture also without using patching, and are able to outperform byte-level transformer models in a FLOP controlled setting at the 350M parameter scale in terms of bits-per-byte on several datasets.

Patching-based approaches: The effective use of patching can bring down the otherwise inflated number of FLOPs expended by byte-level LLMs while potentially retaining performance, and many works demonstrated initial successes at a small scale of model size and number of training bytes. Nawrot et al. [2022] experiment with static patching based downsampling and upsampling and develop the hourglass transformer which outperforms other byte-level baselines at the 150M scale. Nawrot et al. [2023] further improve this with the help of dynamic patching schemes, including a boundary-predictor that is learned in an end-to-end fashion, a boundary-predictor supervised using certain tokenizers, as well as an entropy-based patching model similar to BLT, and show that this approach can outperform the vanilla transformers of the time on language modeling tasks at a 40M parameter scale on 400M tokens. Lester et al. [2024] investigate training on sequences compressed using arithmetic coding to achieve compression rates beyond what BPE can achieve, and by using a equal-info windows technique, are able to outperform byte-level baselines on

language modeling tasks, but underperform subword baselines.

Our work draws inspiration and is most closely related to MegaByte [Yu et al., 2023], which is a decoder only causal LLM that uses a fixed static patching and concatenation of representations to convert bytes to patches, and uses a local model on the decoder side to convert from patches back into bytes. They demonstrate that MegaByte can match tokenizer-based models at a 1B parameter scale on a dataset of 400B bytes. We ablate MegaByte in all our experiments and find that static patching lags behind the current state-of-the-art compute optimally trained tokenizer based models in a FLOP controlled setting and we demonstrate how BLT bridges this gap. Slagle [2024] make the same observation about MegaByte and suggest extending the static patching method to patching on whitespaces and other space-like bytes, and also add a local encoder model. They find improvements over tokenized-based transformer models in a compute controlled setting on some domains such as Github and arXiv at the 1B parameter scale. We also report experiments with this model, and show that further architectural improvements are needed to scale up byte-level models even further and truly match current state-of-the-art token-based models such as Llama 3.

2.9 Limitations and Future Work

In this work, for the purposes of architectural choices, we train models for the optimal number of steps as determined for Llama 3 [Dubey et al., 2024]. However, these scaling laws were calculated for BPE-level transformers and may lead to suboptimal (data, parameter sizes) ratios in the case of BLT. We leave for future work the calculation of scaling laws for BLT potentially leading to even more favorable scaling trends for our architecture. Additionally, many of these experiments were conducted at scales upto 1B parameters, and it is possible for the optimal architectural choices to change as we scale to 8B parameters and beyond, which may unlock improved performance for larger scales.

Existing transformer libraries and codebases are designed to be highly efficient for tokenizer-based transformer architectures. While we present theoretical FLOP matched experiments and also use certain efficient implementations (such as FlexAttention) to handle layers that deviate from the vanilla transformer architecture, our implementations may yet not be at parity with tokenizer-based models in terms of wall-clock time and may benefit from further optimizations.

While BLT uses a separately trained entropy model for patching, learning the patching model in an

end-to-end fashion can be an interesting direction for future work. In Section 2.6.2, we present initial experiments showing indications of success for "byte-ifying" tokenizer-based models such as Llama 3 that are trained on more than 10T tokens, by initializing and freezing the global transformer with their weights. Further work in this direction may uncover methods that not only retain the benefits of bytefying, but also push performance beyond that of these tokenizer-based models without training them from scratch.

2.10 Conclusion

This chapter presents the Byte Latent Transformer (**BLT**), a new architecture that redefines the conventional dependency on fixed-vocabulary tokenization in large language models. By introducing a dynamic, learnable method for grouping bytes into patches, BLT effectively allocates computational resources based on data complexity, leading to significant improvements in both efficiency and robustness. Our extensive scaling study demonstrates that BLT models can match the performance of tokenization-based models like Llama 3 at scales up to 8B and 4T bytes, and can trade minor losses in evaluation metrics for up to 50% reductions in inference FLOPs. Furthermore, BLT unlocks a new dimension for scaling, allowing simultaneous increases in model and patch size within a fixed inference budget. This new paradigm becomes advantageous for compute regimes commonly encountered in practical settings. While directly engaging with raw byte data, BLT also improves the model's ability to handle the long-tail of data, offering significant improvements in robustness to noisy inputs and a deeper understanding of sub-word structures. Overall, these results position BLT as a promising alternative to traditional tokenization-based approaches, providing a scalable and robust framework for more efficient and adaptable scaling of LLMs.

	Llama 3 (1T tokens)	Llama 3.1 (16T tokens)	BLT (1T tokens)
HellaSwag Original	79.1	80.7	**80.6**
HellaSwag Noise Avg.	56.9	64.3	**64.3**
- AntSpeak	45.6	61.3	**57.9**
- Drop	53.8	57.3	**58.2**
- RandomCase	55.3	65.0	**65.7**
- Repeat	57.0	61.5	**66.6**
- UpperCase	72.9	76.5	**77.3**
Phonology-G2P	11.8	18.9	**13.0**
CUTE	27.5	20.0	**54.1**
- Contains Char	0.0	0.0	**55.9**
- Contains Word	55.1	21.6	**73.5**
- Del Char	34.6	34.3	**35.9**
- Del Word	**75.5**	84.5	56.1
- Ins Char	7.5	0.0	**7.6**
- Ins Word	**33.5**	63.3	31.2
- Orthography	43.1	0.0	**52.4**
- Semantic	65	0.0	**90.5**
- Spelling	1.1	-	**99.9**
- Spelling Inverse	30.1	3.6	**99.9**
- Substitute Char	0.4	1.2	**48.7**
- Substitute Word	16.4	6.8	**72.8**
- Swap Char	2.6	2.4	**11.5**
- Swap Word	20.1	4.1	**21**

Table 2.3: We compare our 8B BLT model to 8B BPE Llama 3 trained on 1T tokens on tasks that assess robustness to noise and awareness of the constituents of language (best result bold). We also report the performance of Llama 3.1 on the same tasks and underline best result overall. BLT outperforms the Llama 3 BPE model by a large margin and even improves over Llama 3.1 in many tasks indicating that the byte-level awareness is not something that can easily be obtained with more data.

Language	Language → English		English → Language	
	Llama 3	BLT	Llama 3	BLT
Arabic	22.3	24.6	10.4	8.8
German	41.3	42.0	29.8	31.2
Hindi	20.7	20.9	7.8	7.2
Italian	34.0	33.9	24.4	26.2
Vietnamese	31.2	31.0	28.4	23.7
Thai	17.9	18.1	10.5	7.7
Armenian	1.7	6.3	0.6	0.9
Amharic	1.3	3.1	0.4	0.5
Assamese	2.7	5.4	0.8	1.6
Bengali	4.7	12.7	1.7	4.1
Bosnian	36.0	37.3	16.9	19.6
Cebuano	18.2	20.6	5.8	9.1
Georgian	1.7	7.4	1.0	2.5
Gujarati	2.0	5.8	1.0	2.2
Hausa	5.75	5.9	1.2	1.3
Icelandic	16.1	17.9	4.8	5.3
Kannada	1.6	3.9	0.7	1.7
Kazakh	5.6	7.0	1.0	2.6
Kabuverdianu	20.3	20.9	5.1	6.8
Khmer	4.4	9.5	0.8	0.8
Kyrgyz	4.6	5.1	0.9	2.0
Malayalam	1.8	3.5	0.7	1.4
Odia	1.6	2.7	0.8	1.1
Somali	5.0	5.0	1.1	1.4
Swahili	10.1	12.0	1.4	2.3
Urdu	9.3	9.5	2.0	1.4
Zulu	4.7	5.0	0.6	0.5
Overall Average	12.1	**14.0**	5.9	**6.4**

Table 2.4: Performance of 8B BLT and 8B Llama 3 trained for 1T tokens on translating into and from six widely-used languages and twenty one lower resource languages with various scripts from the FLORES-101 benchmark [Goyal et al., 2022a].

	Llama 3 8B (220B tokens)	BLT 8B (220B tokens)	BLT from Llama 3.1 8B (220B tokens)	Llama 3.1 8B (15T tokens)
Arc-E	67.4	66.8	66.6	83.4
Arc-C	40.4	38.8	45.8	55.2
HellaSwag	71.2	72.2	76.1	80.7
PIQA	77.0	78.2	77.4	80.7
MMLU	26.5	25.2	63.7	66.3
MBPP	11.8	10.0	38.2	47.2
HumanEval	9.2	7.3	34.2	37.2

Table 2.5: Initializing the global transformer model of BLT from the non-embedding parameters of Llama 3 improves performance on several benchmark tasks. First three models trained on the Llama 2 data for compute-optimal steps.

	Llama 3 BPE	Space Patching BLT	Entropy Patch Size 4 BLT
Arc-E	67.4	67.2	68.9
Arc-C	40.5	37.6	38.3
HellaSwag	71.3	70.8	72.7
PIQA	77.0	76.5	77.6

Table 2.6: Benchmark evaluations of two patching schemes for 8b BLT models and BPE Llama 3 baseline. These are compute-optimal models trained on the Llama 2 data.

			BPB			
XAtt. Dec.	XAtt. Enc.	Pool Init	Wiki	CC	Github	Train Dist
-	All Layers	False	0.830	0.915	**0.442**	0.891
-	Last Layer	False	0.836	0.906	0.447	0.886
-	-	-	0.833	0.892	0.446	0.866
First Layer	Last Layer	True	0.825	0.883	0.443	0.861
All Layers	Last Layer	True	**0.823**	0.871	0.443	0.846
All Layers	All Layers	True	0.828	**0.868**	0.443	**0.844**

Table 2.7: Ablations on the use of Cross Attention for a 1B BLT model trained on 100B bytes.

			BPB			
Ngram	Ngram Voc	Total Voc	Wiki	CC	Github	Train Dist
-	-	-	0.892	0.867	0.506	0.850
6,7,8	100k	300k	0.873	0.860	0.499	0.842
6,7,8	200k	600k	0.862	0.856	0.492	0.838
3,4,5	100k	300k	0.859	0.855	0.491	0.837
6,7,8	400k	1M	0.855	0.853	0.491	0.834
3,4,5	200k	600k	0.850	0.852	0.485	0.833
3,4,5,6,7,8	100k	600k	0.850	0.852	0.486	0.833
3,4,5	400k	1M	0.844	0.851	0.483	0.832
3,4,5,6,7,8	200k	1M	0.840	0.849	0.481	0.830
3,4,5,6,7,8	400k	2M	**0.831**	**0.846**	**0.478**	**0.826**

Table 2.8: Ablations on the use of n-gram hash embedding tables for a 1B BLT model trained on 100B bytes. We find that hash n-gram embeddings are very effective with very large improvements in BPB. The most significant parameter is the per-ngram vocab size and that smaller ngram sizes are more impactful than larger ones.

Ngram Embeds	Enc Layers	Dec Layers	BPB
False	1	9	0.850
False	5	5	0.843
True	5	5	0.844
True	3	7	0.824
True	1	9	0.822

Table 2.9: When paired with hash n-gram embeddings, a light-weight local encoder is sufficient. More layers can then be allocated to the decoder for the same cost.

Chapter 3

SOCRATIC Pretraining: Question-Driven Pretraining for Controllable Summarization

The architecture of a language model can have significant impact on scaling efficiency during pretraining as demonstrated in chapter 2. However, the data it is trained on is equally important. Significant effort in recent research on LLM scaling trends focuses on improving data quality for LLM training [Li et al., 2024]. Part of the challenge is that we are rapidly approaching the use of the entire web data especially the portions considered to include higher quality data. In this chapter, we explore the possibility of augmenting web data with careful synthetic generations. In this chapter, we target the mid-training or continued pretraining stage of the LLM development pipeline and demonstrate how synthetic data augmentation can reduce task-specific data requirements by half and improve adherence to task specifications in later tuning stages. Overall, we find that data quality is central to improving the scaling efficiency of language models and reduce the reliance on expensive and limited human authored data. This chapter includes materials originally published in Pagnoni et al. [2023].

3.1 Introduction

Summarization systems are designed to help users navigate large amounts of information [Edmunds and Morris, 2000], but often fail to meet the unique needs of different users, especially for long documents.

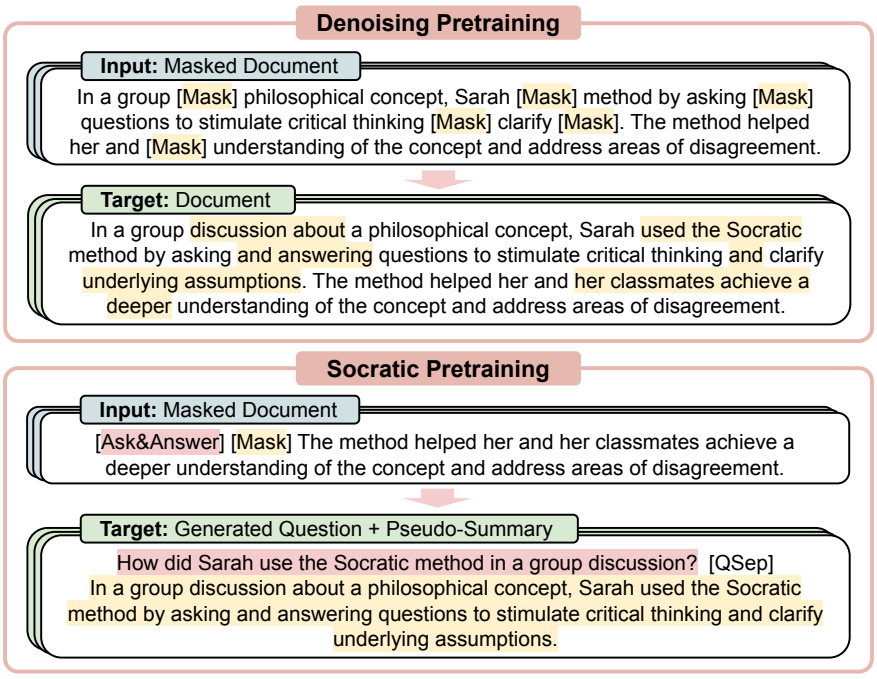

Figure 3.1: Our SOCRATIC pretraining compared to denoising. We mask important sentences in unlabeled input documents and train the model to generate both **questions** and **pseudo-summaries** as their answers.

Recent research has explored ways to make summarization systems more controllable [Bornstein et al., 1999; Leuski et al., 2003] by allowing users to input queries or control sequences such as keywords [He et al., 2022a], questions [Zhong et al., 2021a], entity chains [Narayan et al., 2021], or question-answer pairs [Narayan et al., 2022].

A challenge shared by all of the mentioned approaches is the absence of abundant labeled data. Currently available datasets for training these systems are the result of expensive annotation efforts [Zhong et al., 2021a; Kulkarni et al., 2020; Wang et al., 2022a] with only hundreds to a few thousand query-document pairs, with the same document often being repeated. This translates into poor adherence of generated summaries to user-provided queries, particularly when these are finegrained plans. Recent work demonstrates the benefits of tailoring the pretraining objective to downstream task characteristics, especially where training data is difficult to obtain in large quantities like factuality-focused and multi-document summarization [Wan and Bansal, 2022; Xiao et al., 2022b]. In controllable summarization, summaries are grounded by queries, so designing an objective for the task requires introducing realistic queries in unlabeled data in a scalable manner.

This chapter introduces SOCRATIC pretraining, an unsupervised pretraining objective for language models that is specifically designed for controllable summarization. It is inspired by the Socratic method and aims to facilitate the identification of relevant content and ensure that the generated summary faithfully responds to the user query. During SOCRATIC pretraining (see Figure 3.1) the language model is trained to generate relevant questions based on an input document and then answer them, bringing finegrained controllability to model pretraining which translates to better adherence to user queries.

SOCRATIC pretraining only relies on unlabeled data and a question generation system and outperforms pre-finetuning approaches relying on additional supervised data [Aghajanyan et al., 2021; Wei et al., 2021; Fabbri et al., 2021a]. In this chapter, we demonstrate the effectiveness of the SOCRATIC objective through pretraining adaptation, where a language model is further pretrained with the SOCRATIC objective before finetuning on task-specific labeled data.

In summary, the contributions of this chapter are as follows[1]:

- We introduce the SOCRATIC pretraining objective for controllable summarization to improve adherence to user-specified queries or plans, both high-level and finegrained.
- We show that SOCRATIC pretraining performs well across domains, control strategies, and achieves state-of-the-art performance on two datasets.
- We perform ablations on our approach showing that SOCRATIC pretraining cuts labeled data requirements in half.

3.2 Related Work

Task-Specific Pretraining Adaptation Current state-of-the-art methods in abstractive summarization apply a two-step approach where models are first pretrained on large corpora of text with task-agnostic variations of the text denoising objective and next finetuned on labeled examples from the target task [Lewis et al., 2020; Raffel et al., 2020a].

However, in tasks where labeled data is scarce, task-specific pretraining objectives have been shown to provide significant benefits. Recent work adapted language models to summarize multiple documents [Xiao

[1]The supporting code is available at `https://github.com/salesforce/socratic-pretraining`

et al., 2022b], produce more factual summaries [Wan and Bansal, 2022], or plan with entity chains [Narayan et al., 2021]. We build on these methods, focusing on the downstream task of controllable summarization.

Other studies demonstrate the effect of continued pretraining [Gururangan et al., 2020] and pre-finetuning [Aghajanyan et al., 2021; Wei et al., 2021; Fabbri et al., 2021a] on downstream task adaptation. These either continue training with the same objective on data in the downstream task domain or perform multitask learning using labeled data. In this chapter, we demonstrate the benefits of language model adaptation with a task-specific pretraining objective without additional supervised data and show that these benefits are consistent and statistically significant in low-resource settings like query-focused summarization (QFS).

Controllable Summarization Controllable text generation [Hu et al., 2017] aims to control properties of the generated text including style [Kumar et al., 2021], length, or content [Fan et al., 2018; He et al., 2022a]. Approaches for content control vary according to the type of control: keywords [He et al., 2022a], entities [Narayan et al., 2021], questions [Vig et al., 2022], factoid question-answer pairs (also called QA blueprints) [Narayan et al., 2022]. As opposed to methods like GSum [Dou et al., 2021], which insert control tokens on the encoder side, we focus on decoder-based methods which do not require re-encoding the document when the control sequences are updated. In summarization, these controls can broadly indicate the information to summarize, like the questions in query-focused summarization, or provide a detailed plan of the text to be generated, like the entity chains. While these types of control are not typically studied together we show that our SOCRATIC pretraining provides benefits across the board for both high-level and finegrained queries and plans.

Learning with Questions Inspired by the Socratic method, recent literature in education theory shows students generate questions as a way of learning [Rosenshine et al., 1996; Aflalo, 2021], hinting at the potential benefits that could derive from incorporating questions during model training. Previous work shows that question-answer pairs, both generated [Du et al., 2017; Alberti et al., 2019; Ko et al., 2021; Murakhovs'ka et al., 2022; Chakrabarty et al., 2022] and from the web [Narayan et al., 2020], can provide useful training signal for pretrained encoders [Jia et al., 2022] as well as question generation and abstractive summarization systems [Narayan et al., 2022]. Our SOCRATIC objective builds on these observations and is designed to improve sequence-to-sequence model pretraining for more controllable summarization systems. Similar to

information-seeking Dialogue Inpainting [Dai et al., 2022], SOCRATIC pretraining extracts questions from unlabeled data focusing on higher-level questions, whose answers are full sentences, instead of factoid QA pairs.

3.3 SOCRATIC Pretraining

During SOCRATIC pretraining, the model takes as input a document with important sentences masked and is trained to generate questions about the masked content and produce the mask itself. As seen in Figure 3.1, SOCRATIC pretraining is formulated as a sequence-to-sequence task and consists of two steps 1) important content is selected from unlabeled documents to be masked, and 2) a question-generation system is applied to produce questions about the selected content. The question augmentation component trains the model to produce summaries grounded to questions and allows for controllability as the end-user can prompt the model decoder with new questions during inference. We describe both steps below.

3.3.1 Content Selection

Selecting important content is essential for the model to learn to generate salient questions and summaries. In SOCRATIC pretraining, this content selection is done using the PEGASUS-style Gap Sentence Generation (GSG) objective [Zhang et al., 2020a], which we now briefly describe. Sentences with the highest self-Rouge with the document are selected for masking, ensuring that there is high information overlap with the rest of the document. The selected sentences, concatenated, produce a pseudo-summary of the document. As in PEGASUS, a Gap Sentence Ratio (GSR) of 45% is used, meaning that 45% of the sentences in the document are selected to appear in the target pseudo-summary. To help the model learn to copy, 80% of these sentences are masked and 20% are kept unmasked in the input document. Documents and summaries are truncated to 512 and 256 tokens.

3.3.2 Question Augmentation

After selecting the pseudo-summary, a question generation (QG) system is applied to obtain a question from each sentence of the pseudo-summary. The QG system takes as input one of the selected sentences at a time

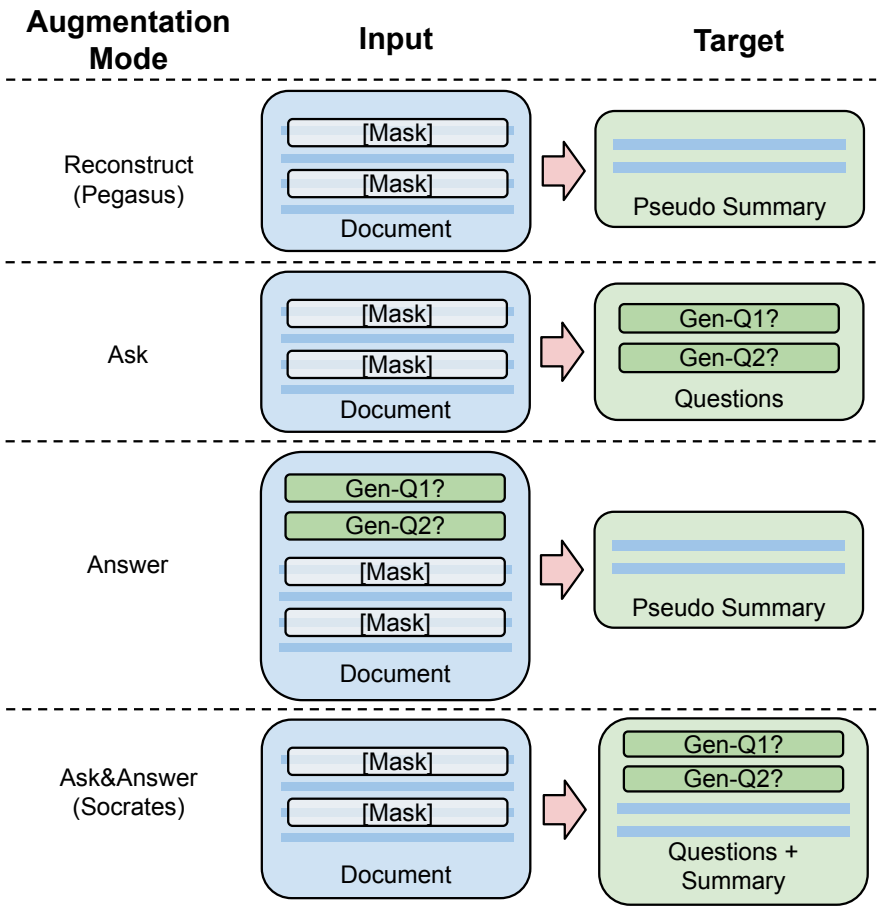

Figure 3.2: SOCRATIC augmentation modes vs. Pegasus.

and the unmasked document as context. We apply MixQG [Murakhovs'ka et al., 2022], a state-of-the-art QG system.

The choice to generate a question for each selected sentence, as opposed to each entity or the entire summary, is driven by three reasons. First, sentences in the pseudo-summary are selected from across the document and generally lack coherence, so there is no single query they collectively answer. Second, current QG systems are not trained to produce paragraph-level questions. Third, entity-level questions are often simple paraphrases of the answer sentence and are uncommon in QFS datasets.

Questions whose answers are full sentences, therefore, offer a compromise in terms of the complexity of the question and the coherence of the answer. We refer to these sentence-level questions as content-questions as they tend to ask about the content of the document instead of specific entities.

3.3.3 Training Objective

After obtaining the questions, there are multiple ways to introduce them in the training objective either in the input or in the target text. As seen in Figure 3.2, we experiment with three modes on top of the base GSG objective:

- Reconstruct. The reconstruct mode is the default GSG mode where no questions are introduced. The masked document is the input and the pseudo-summary is the target text. We provide this mode as a baseline for our approach.

- Ask. Given the masked document as input, the model is trained to only predict the questions about the masked sentences. This is the only mode where the target text does not include the pseudo-summary. With this mode, the model is trained to predict which questions can be asked in a given context.

- Answer. Here, the questions are prepended to the masked input document while the target text remains the pseudo-summary. This mode is similar to how queries are introduced to the model during query-focused summarization and should help the model learn to respond to user-provided queries. However, this mode forgoes content planning as each generated sentence corresponds to one of the questions prepended to the input.

- Ask&Answer. This mode combines benefits from both Ask and Answer modes. The model is tasked to first generate questions about the document and then, conditioning on both the document and the generated questions, the pseudo-summary. The model conditions on the generated questions in the decoder. This mode can be seen as first generating a finegrained plan for the pseudo-summary and then the pseudo-summary itself.

Like Tay et al. [2023], we prepend special tokens `<ask>`, `<answer>`, and `<ask&answer>` to the input document to specify the augmentation mode, and the `<qsep>` token to separate the generated questions from the target pseudo-summary.

3.4 Experimental Setup

We describe the experimental setup that we use to study SOCRATIC pretraining along with empirical studies justifying our design decisions.

3.4.1 Model Architecture

The SOCRATIC objective can be applied to any sequence-to-sequence language model irrespective of its specific architecture. In our experiments, we choose BART-large [Lewis et al., 2020], as the starting point for SOCRATIC pretraining adaptation. Following previous work on pretraining adaption for summarization, we pick BART over PEGASUS for its smaller size without performance compromises on summarization benchmarks and its more general-purpose pretraining objective. BART is also the underlying model in the SegEnc [Vig et al., 2022] architecture, which achieved state-of-the-art performance on QMSum, outperforming models such as LongT5 [Guo et al., 2022].

Instead of pretraining the language model from scratch, we demonstrate the effectiveness of the proposed objective through what we call pretraining adaptation, where a generic language model is further pretrained with the SOCRATIC objective before being finetuned on task-specific labeled data. Although we introduce a new term for this training phase, pretraining adaptation was recently employed to evaluate task-specific pretraining objectives for factuality and multi-document summarization [Wan and Bansal, 2022; Xiao et al., 2022b].

After SOCRATIC pretraining adaptation, the resulting model is used to initialize the SegEnc architecture, which is then finetuned on labeled data from downstream tasks. Pretraining and finetuning hyperparameter details are available in 7.1.2.

3.4.2 Pretraining Corpus

We experiment with three different corpora, two of which are part of the Pile [Gao et al., 2020].

- OpenWebText2 is a web-scraped dataset inspired by WebText [Radford et al., 2019] that uses Reddit upvotes of outgoing links as a proxy for page quality. Raffel et al. [2020a] found this dataset to work well for summarization pretraining.

- Books3 is a collection of both fiction and non-fiction books. We explore this data because our downstream tasks involve the short story and dialogue domains, and Csaky and Recski [2021] show books can be a good source of dialogue data.

- UnDial [He et al., 2022b] We also explore using a dialogue corpus. As there are only two speakers in each dialogue in UnDial, we use a simple rule-based system to convert dialogues to third person. The pseudo-summary and related questions are then expressed in the third person while the input remains in the original dialogue format.

3.4.3 Downstream Tasks

To determine whether SOCRATIC pretraining improves model initialization for finetuning on controllable summarization, we test on two downstream datasets for query-focused, long-document summarization: QMSum and SQuALITY (dataset statistics can be found in 7.1.1). We focus on long document datasets as a challenging and practical testbed for controllable summarization methods.

QMSum. QMSum is a benchmark for query-based, multi-domain meeting summarization [Zhong et al., 2021a]. The dataset consists of 1,808 query-summary pairs over 232 meetings, including product, academic, and parliamentary meetings.

SQuALITY. SQuALITY is a dataset for query-based short stories summarization [Wang et al., 2022a]. The dataset is composed of 625 examples over 100 stories with four long reference summaries per document-question pair.

3.4.4 Evaluation Protocol

We apply the standard Rouge [Lin, 2004] and BERTScore [Zhang et al., 2020b] metrics to compare model generations with reference summaries on downstream finetuning tasks. In SQuALITY, we use the same procedure as the dataset authors to incorporate multiple references by taking the maximum score over the reference summaries. We also conduct a human evaluation study to ensure the variations between models are meaningful to users. Details on the setup can be found in 7.1.4.

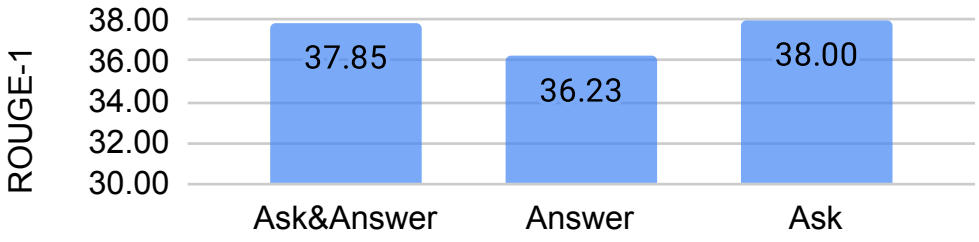

Figure 3.3: Comparison of question augmentation modes.

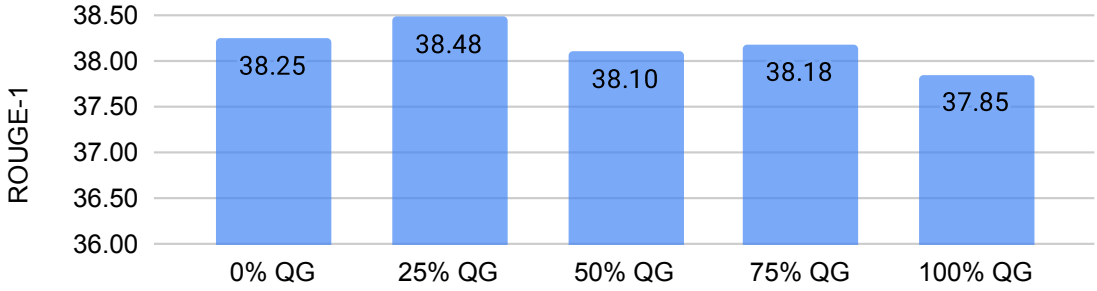

Figure 3.4: Comparison of QG augmentation proportions.

3.5 SOCRATIC Pretraining Ablations

In this section, we corroborate our design choices with ablation studies of the components of SOCRATIC pretraining. Similar to Zhang et al. [2020a] and Raffel et al. [2020a], to save time and resources, we conduct the ablations of the objective on a small scale by restricting the pretraining adaptation to 1M documents from the OpenWebText2 corpus and then finetuning it on the full downstream task datasets. We report the mean over five randomly initialized finetuning runs on the validation set.

Question Augmentation Modes In Figure 3.3, we compare the performance of the three approaches for incorporating generated questions in the SOCRATIC objective. The Ask and Ask&Answer perform similarly while Answer lags behind. This is in line with our hypothesis that learning which questions are relevant in a given context is a useful training signal for the model. The Ask&Answer mode also grounds the pseudo-summary generation in a sequence of finegrained questions. Therefore, it is chosen to be used in SOCRATIC pretraining.

(a) QMSum **(b)** Squality

Figure 3.5: Effect of the pretraining corpus (dev set).

Question Augmentation Proportion Incorporating questions with the Ask&Answer mode in each pretraining example could bias the model to always start by generating questions. We hypothesize that combining the Reconstruct mode with the Ask&Answer mode could alleviate this bias. In Figure 3.4, we find that introducing questions in 25% of pretraining examples leads to the best performance and use this proportion when scaling the pretraining adaptation.

Pretraining Corpus Selection In Figure 3.5, we find that the choice of pretraining corpus has a small but consistent effect on the performance of the SOCRATIC pretrained model on downstream tasks. The Books3 corpus performs best both on QMSum and SQuALITY. The dialogue corpus offers a slight advantage over OpenWebText2 on QMSum, a dialogue summarization task, while the opposite is true for SQuALITY. As a result, the full Books3 corpus, consisting of 30M training instances, is used in further experiments.

3.6 Query Focused Summarization Results

We scale the SOCRATIC pretraining adaptation based on the findings of the previous ablation and evaluate its downstream effects on query-focused summarization. Unless specified, the results in this section are averaged over five randomly initialized finetuning runs on the downstream tasks.

In Table 3.1, we compare the effect of SOCRATIC pretraining to other pretraining strategies on QMSum and SQuALITY. We obtain an improvement of +1.01 and +0.53 Rouge-1, respectively, surpassing even the use of additional supervision from the related dataset WikiSum in Vig et al. [2022] and achieving new state-of-the-art results. These improvements are validated by a human study reported in Figure 3.6 and showing that SOCRATIC SegEnc performs better than the baselines in 59-65% of instances. Details of the human

Model	Rouge1	Rouge2	RougeL	BS-R
QMSum				
BART-LS [Xiong et al., 2023]	37.90	12.10	33.10	-
BART-Large SegEnc	37.05	13.04	32.62	87.44
+ *WikiSum Pre-Finetuning*	*37.80*	*13.43*	*33.38*	-
+ BART Pret. 1M	36.64	12.44	31.94	86.94
+ SOCRATIC Pret. 1M	37.46	13.32	32.79	87.54
+ PEGASUS Pret.	37.29	13.30	32.70	87.48
+ SOCRATIC Pret.	**38.06**	**13.74**	**33.51**	**87.63**
Squality				
LED	27.7	5.9	17.7	-
PEGASUS	38.2	9.0	20.2	-
BART	40.2	10.4	20.8	-
BART + DPR	41.5	11.4	21.0	-
Human	46.6	12.5	22.7	-
BART-Large SegEnc	45.68	14.51	22.47	85.86
+ PEGASUS Pret.	45.78	14.43	**22.90**	85.94
+ SOCRATIC Pret.	**46.31**	**14.80**	22.76	**86.0**4

Table 3.1: Results on QMSum and SQuALITY with pretraining on Books3. Baselines from Vig et al. [2022] and Wang et al. [2022a] respectively. 1M indicates that 1M pretraining instances are used.

evaluation are found in 7.1.4.

3.6.1 Disentangling the Effect of Questions

The main baseline for SOCRATIC pretraining is the PEGASUS style GSG pretraining. We therefore perform a pretraining adaptation of BART-large with the GSG objective on the full Books3 corpus. In Table 3.1, we observe that GSG pretraining on the full Books3 corpus improves by +0.24 Rouge-1 over the BART SegEnc model. However, with the SOCRATIC objective, 1M examples from Books3 (1/30 of the full corpus) are sufficient to surpass GSG pretraining, with a +0.41 Rouge-1 improvement over BART SegEnc. This indicates that GSG pretraining, tailored to generic summarization, is only marginally helpful in tasks where summaries have to answer user-provided queries. In addition, increasing the corpus for SOCRATIC pretraining to the entire Books3 corpus further improves the performance by +0.60 Rouge-1 on QMSum, showing that the benefits of the pretraining objective do not saturate early and that the model continues to improve

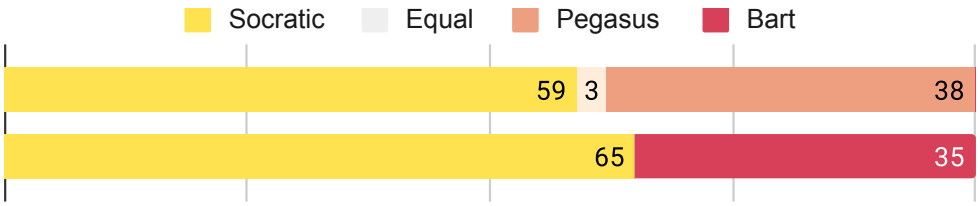

Figure 3.6: Human annotators' preferences on QMSum.

with additional SOCRATIC pretraining.

We also compare to BART-LS, an orthogonal approach that tailors BART's architecture, pretraining corpus, and objective to long documents [Xiong et al., 2023]. While our approaches are complementary, we outperform BART-LS on QMSum by +1.64 Rouge-2. This confirms our hypothesis that grounding generations in control queries in SOCRATIC pretraining is beneficial in controllable summarization, even more so than better long document modeling.

3.6.2 Comparing to Continued Pretraining

Gururangan et al. [2020] show that language models can be successfully adapted to the task domain by continuing to pretrain them in the new domain. This raises the question of whether improvements due to SOCRATIC pretraining are simply due to a better affinity of the pretraining corpus to the task domain. To answer this question, we perform continued pretraining[2] on a 1M subset of the Books3 corpus and next finetune the model on QMSum. Table 3.1 shows that continued pretraining slightly hurts Rouge-1 performance. In comparison, performing SOCRATIC pretraining on the same corpus improves performance by +0.41 Rouge-1. This observation rules out that improvements achieved through SOCRATIC pretraining are simply due to improved domain adaptation.

3.6.3 Comparing to Pre-Finetuning

Transferring information from related tasks is another approach to adapt generic models to specific tasks [Aghajanyan et al., 2021]. We show in Table 3.1 that SOCRATIC pretraining outperforms even the best pre-finetuned BART SegEnc model, which uses additional supervision from the WikiSum dataset [Liu et al., 2018]. This transfer dataset was selected from a wide range of relevant summarization datasets tested by Vig

[2]For consistency, we use Fairseq to pretrain BART-large

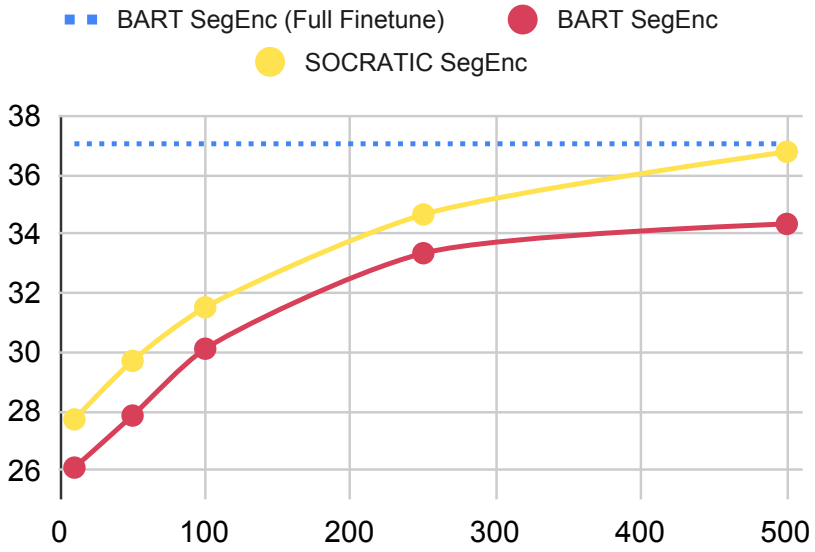

Figure 3.7: Few-shot performance on QMSum test set.

et al. [2022]. Crucially, we note that transfer learning, like pre-finetuning, is orthogonal to our line of work which operates on the pretraining side. We believe that SOCRATIC can therefore be used in combination with pre-finetuning to further boost performance.

3.6.4 General vs. Specific Summaries

Both QMSum and Squality datasets contain a substantial portion of general summaries (12.5-20%) that aim to summarize the entire document in addition to those answering more specific queries. We find that our approach improves in both cases (+0.98 and +0.28 ROUGE-1 on QMSum in general and specific queries respectively). This shows that SOCRATIC pretraining improves models intended to perform a combination of general-purpose and query-focused summarization. In addition, with users increasingly interacting with language models through prompts to perform different tasks, the query-focused datasets we evaluate on become realistic testbeds for NLP systems that aim to perform well across tasks.

3.6.5 Few-Shot Finetuning

To show that SOCRATIC pretraining alleviates the need for labeled downstream task data, we study the few-shot learning performance of SOCRATIC and BART SegEnc models. We perform one finetuning run for

each model on each subset of the task data. In Figure 3.7, we show that with half the QMSum examples, SOCRATIC SegEnc achieves the same performance as finetuning BART SegEnc on all of QMSum. We believe that bringing SOCRATIC pretraining closer to the downstream task of query-focused summarization lets the models learn from fewer downstream task examples.

3.7 Finegrained Planning Results

In this section, we evaluate the effect of SOCRATIC pretraining on the adherence to user-provided finegrained control sequences. In these experiments, the same SOCRATIC pretrained model is finetuned on task-specific data with various control strategies.

3.7.1 Going Beyond High-Level Questions

The queries found in QMSum and SQuALITY are only one format to encode user intent. Previous research explored other control strategies like keywords He et al. [2022a], entity chains [Narayan et al., 2021], or factoid question-answer pairs [Narayan et al., 2022]. As seen in Figure 3.8, these strategies offer a more finegrained level of control over the summaries as they operate at the sentence level. Reference control sequences are not available for QMSum and SQuALITY so we generate them automatically from reference summaries. In the summarization literature, such control sequences are often modeled as intermediate plans generated before the summaries [Narayan et al., 2022; He et al., 2022a]. In these cases, given the input X, the model first generates the detailed plan for the summary B from $P(B|X)$, then generates the summary Y conditioning on the plan and the input x from $P(Y|B, X)$. Even if the plan B is initially generated by the model, a user can control the summary by altering the plan. In practice, we experiment with three different planning strategies.

- Content questions. For each sentence in the reference summary, we generate a question using the MixQG system while giving the full summary as context. These are similar to the questions that we use in our SOCRATIC pretraining. The sentence-level questions are then concatenated into a single plan for the summary. To our knowledge, we are the first to propose using content questions as finegrained plans for summaries.

Original Text: In a group discussion about a philosophical concept, Sarah used the Socratic method by asking and answering questions to stimulate critical thinking and clarify underlying assumptions. The method helped her and her classmates achieve a deeper understanding of the concept and address disagreements. Sarah looked forward to continuing to use it in her studies.

Content Questions (Ours): How did Sarah use the Socratic method? What were the benefits of the Socratic method? What did Sarah think of the method?

Keywords: Group discussion | Sarah | Socratic method | questions | thinking | assumptions || method | classmates | understanding | disagreement || studies

Blueprint QA: What type of discussion did Sarah have about a philosophical concept? Group discussion | Who used the Socratic method? Sarah | What method did Sarah use to stimulate critical thinking? Socratic method | What did Sarah ask in the Socratic method? questions | What did Sarah clarify in the Socratic method? assumptions ...

Figure 3.8: Comparison of finegrained control strategies.

- QA blueprint. We reimplement the recently proposed text plan in the form of a sequence of question-answer (QA) pairs [Narayan et al., 2022]. First, all noun phrase answers are extracted from the reference. Then, a QG system generates questions answered by each noun phrase. The QA pairs are then filtered using round-trip consistency, rheme, and coverage criteria. The final plan consists of the concatenation of the remaining QA pairs.

- Keywords. We use keywords extracted from each sentence of the reference summary. We take the noun-phrase answers from the QA blueprint as keywords and concatenate them with sentence separators into a plan.

		Summary				Control Plan			
Control Strategy	Model	Rouge1	Rouge2	RougeL	BS-R	Rouge1	Rouge2	RougeL	Leven. Edit
Content Questions	BART-Large SegEnc	35.3	11.6	30.7	86.95	42.3	**23.4**	41.6	0.77
	+ PEGASUS Pret.	35.4	11.8	30.9	87.03	41.7	22.9	41.0	0.74
	+ SOCRATIC Pret.	**36.0**	**12.1**	**31.5**	**87.15**	**42.4**	23.2	**41.7**	0.77
Blueprint QA	BART-Large SegEnc	33.5	9.3	29.4	86.62	40.2	15.7	39.2	0.85
	+ SOCRATIC Pret.	**35.4**	**10.0**	**30.6**	**86.89**	**40.7**	**15.9**	**39.6**	0.85
Keywords	BART-Large SegEnc	36.2	12.8	31.4	**87.01**	24.1	9.2	21.3	0.88
	+ SOCRATIC Pret.	**36.9**	**13.2**	**32.1**	**87.01**	**25.0**	**10.0**	**22.1**	0.88

Table 3.2: Results on different control strategies on QMSum (results averaged over five random seeds).

Oracle Strategy	Model	R-1	R-2	R-L	BS-R
Content Questions	BART-Large SegEnc	43.7	18.0	39.0	88.32
	+ SOCRATIC Pret.	**46.8**	**20.3**	**41.7**	**88.92**
Blueprint QA	BART-Large SegEnc	52.9	24.1	46.8	89.63
	+ SOCRATIC Pret.	**56.3**	**26.6**	**49.3**	**90.03**
Keywords	BART-Large SegEnc	45.7	20.2	40.5	88.73
	+ SOCRATIC Pret.	**47.5**	**21.9**	**42.5**	**89.18**

Table 3.3: Performance on the QMSum dataset with various oracle finegrained control strategies.

3.7.2 Comparing Control Strategies

In Table 3.2, we report evaluation metrics for both the model-generated summaries and plans.

We find that with all three control strategies, SOCRATIC pretraining provides a consistent improvement over the vanilla BART model and the PEGASUS pretraining on both the generated finegrained plan and summary. On the planning side, there is a small but consistent improvement, up to +0.9 Rouge-1 with keyword chain control, indicating that the model has improved planning abilities. On the summarization side, we find a more significant improvement with up to +1.9 Rouge-1 with blueprint QA control. We attribute this to a combination of improved planning and execution ability of the model from SOCRATIC pretraining.

With respect to control strategy performance, we find that our content questions obtain the highest Rouge scores (42.4 Rouge-1), outperforming keyword chains with only 25.0 Rouge-1. Despite the keyword plan having low overlap with the reference, it results in good summarization performance, so it is unclear whether the model using keyword chains learns the right correspondence between plan and summary. Moreover, the generated keyword chain would need heavier editing to obtain the reference plan compared to the content question plan (0.88 Levenstein distance compared to 0.77), making them less useful in practice.

Previous work has focused on keyword controls [He et al., 2022a] and fact-oriented questions for text generation [Narayan et al., 2022], but there are inherent limitations with these approaches, which we discuss in detail in 7.1.5.

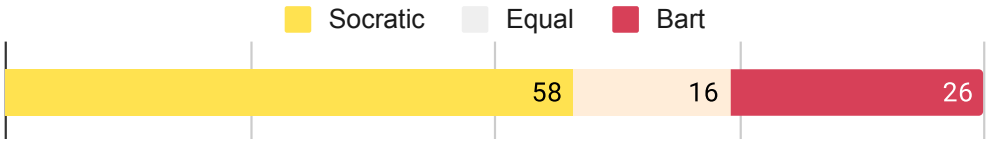

Figure 3.9: Annotators' finegrained planning preferences.

3.7.3 Oracle Questions

Ideally, users can tailor generated summaries with an intervention limited to editing the generated plans. However, this requires strong adherence of generations to the finegrained plans, which we test here with oracle plans. Instead of generating both plan and summary, the system is given the oracle plans automatically extracted from the reference summaries (see 3.7.1). In Table 3.3, we observe a large improvement of +3.1 Rouge-1 over the BART SegEnc baseline. Human annotators confirm that SOCRATIC SegEnc follows oracle finegrained plans better or similarly to the baseline in 74% of instances, shown in Figure 3.9 and described further in 7.1.4. This confirms our hypothesis that SOCRATIC pretraining helps ground the generations to user-provided queries. We attribute these gains to using the Ask&Answer mode, which introduces structure in the pretraining data by using as target text a question plan followed by its pseudo-summary answer. We hypothesize that this structure in pretraining is what helps the model adhere to the planning step more effectively regardless of the control strategy.

3.8 Limitations

Downstream Tasks In this chapter, we focused on long-document summarization as we believe it is the task where controllable summarization is most needed. Future work could investigate the effect of SOCRATIC pretraining on other downstream applications beyond those studied here. To handle long document input we could not use the BART model with SOCRATIC pretraining adaptation directly. Instead, we applied the SegEnc architecture on top of BART. This adaptation of the pretrained model may have dampened some of the few-shot performance of SOCRATIC pretraining. We thus believe that tasks with shorter input documents for which the SegEnc architecture is not necessary would see even greater benefits in the low-resource setting.

Base Model Throughout this chapter, we restricted our analysis to one model architecture the SegEnc architecture with the BART base model. Previous work extensively studied the impact of different architectures for long-document query-focused summarization [Vig et al., 2022]. These primarily differ in how they model long documents. The authors found SegEnc, a simple sliding window adaptation of BART, to perform best on QMSum. While the results presented here are specific to SegEnc and BART, our approach is agnostic to the underlying model architecture and is orthogonal to long-document modeling. We leave it to future work to investigate the effect SOCRATIC pretraining has on other architectures.

Evaluation Metrics As discussed in prior work [Fabbri et al., 2021b; Pagnoni et al., 2021; Gehrmann et al., 2021], there are limitations with the current automated evaluation metrics which do not strongly correlate with human judgments. Our results from these metrics should therefore be interpreted with caution and in combination with the human evaluation we performed to support them. One area in which automated metrics have been reported to perform poorly is factuality. Moreover, current factuality metrics have been designed and tested in the news domain and their performance in the out-of-domain setting (long documents and dialog data) was not systematically evaluated and is hard to interpret [Agarwal et al., 2022]. In this chapter, we therefore choose not to report any factuality metric results.

QG Efficiency We did not optimize the efficiency of the QG component of SOCRATIC pretraining and, consequently, it is computationally expensive. Currently, given equal amounts of resources for QG and pretraining, it takes us about the same time to perform the QG phase and pretraining phase on the same amount of data. We note, however, that in low-resource scenarios, the additional compute can lead to significant benefits, as shown in our results. In addition, we did not experiment with efficient sampling strategies, and believe that improving the efficiency of the QG model inference, for example through model distillation [Hinton et al., 2015], could lead to significant efficiency gains.

Dataset Biases The datasets for pretraining and finetuning used in this chapter are in English and thus mainly represent the culture of the English-speaking populace. Political or gender biases may also exist in the dataset, and models trained on these datasets may propagate these biases. Additionally, the pretrained BART model carries biases from the data it was pretrained on. We did not stress test these models for biases

and request that the users be aware of these potential issues in applying the models presented.

Misuse Potential and Failure Mode When properly used, the summarization models described in this chapter can be time-saving. However, the current model outputs may be factually inconsistent with the input documents, and in such a case could contribute to misinformation on the internet. This issue is present among all current abstractive summarization models and is an area of active research.

3.9 Conclusion

In this chapter, we introduce SOCRATIC pretraining, a question-driven, unsupervised pretraining objective to adapt generic language models to the task of controllable summarization. SOCRATIC pretraining augments web data with synthetically generated questions training the model to generate relevant questions to a given context and then answer them. Our experiments demonstrate the generality of our approach both on query-focused summarization and finegrained controllable summarization. We show that SOCRATIC pretraining outperforms other pretraining and prefinetuning objectives, that it cuts downstream task data requirements in half, and that it works across control strategies and domains. This chapter demonstrates the importance of high quality training signal during mid-training or continued pretraining stages of LLM development. We showed this can have significant reduce the need for scaling expensive and limited human authored task-specific data.

Chapter 4

QLoRA: Efficient Finetuning of Quantized LLMs

Pretraining (see chapter 2) and mid-training (see chapter 3) make LLM broadly knowledgeable, but post-trained makes it *useful*, *safe*, and *affordable*. In this chapter, we present QLoRA-an efficient finetuning method that significantly reduces the memory requirements of post-training and inference of LLMs. By combining 4-bit quantization with parameter efficient finetuning, QLoRA reduces the memory footprint of model finetuning by 15$\times$ without requiring performance tradeoffs over full-precision methods. This also enables an extensive study on supervised finetuning recipes for chatbot performance revealing the importance of quality over quantity of data for downstream performance. This chapter includes materials originally published in Dettmers et al. [2023].

4.1 Introduction

Finetuning large language models (LLMs) is a highly effective way to improve their performance, [Min et al., 2021; Wei et al., 2021; Ouyang et al., 2022; Wang et al., 2022c,b; Liu et al., 2022a] and to add desirable or remove undesirable behaviors [Ouyang et al., 2022; Askell et al., 2021; Bai et al., 2022a]. However, finetuning very large models is prohibitively expensive; regular 16-bit finetuning of a LLaMA 65B parameter model [Touvron et al., 2023] requires more than 780 GB of GPU memory. While recent

Model	Size	Elo
GPT-4	-	1348 ± 1
Guanaco 65B	41 GB	1022 ± 1
Guanaco 33B	21 GB	992 ± 1
Vicuna 13B	26 GB	974 ± 1
ChatGPT	-	966 ± 1
Guanaco 13B	10 GB	916 ± 1
Bard	-	902 ± 1
Guanaco 7B	6 GB	879 ± 1

Table 4.1: Elo ratings for a competition between models, averaged for 10,000 random initial orderings. The winner of a match is determined by GPT-4 which declares which response is better for a given prompt of the the Vicuna benchmark. 95% confidence intervals are shown ($\pm$). After GPT-4, Guanaco 33B and 65B win the most matches, while Guanaco 13B scores better than Bard.

quantization methods can reduce the memory footprint of LLMs [Dettmers et al., 2022a; Dettmers and Zettlemoyer, 2023; Frantar et al., 2022; Xiao et al., 2022a], such techniques only work for inference and break down during training [Wortsman et al., 2023].

We demonstrate for the first time that it is possible to finetune a quantized 4-bit model without any performance degradation. Our method, QLoRA, uses a novel high-precision technique to quantize a pretrained model to 4-bit, then adds a small set of learnable Low-rank Adapter weights [Hu et al., 2021] that are tuned by backpropagating gradients through the quantized weights.

QLoRA reduces the average memory requirements of finetuning a 65B parameter model from >780GB of GPU memory to <48GB without degrading the runtime or predictive performance compared to a 16-bit fully finetuned baseline. This marks a significant shift in accessibility of LLM finetuning: now the largest publicly available models to date finetunable on a single GPU. Using QLoRA, we train the **Guanaco** family of models, with the second best model reaching 97.8% of the performance level of ChatGPT on the Vicuna [Chiang et al., 2023] benchmark, while being trainable in less than 12 hours on a single consumer GPU; using a single professional GPU over 24 hours we achieve 99.3% with our largest model, essentially closing the gap to ChatGPT on the Vicuna benchmark. When deployed, our smallest **Guanaco** model (7B parameters) requires just 5 GB of memory and outperforms a 26 GB Alpaca model by more than 20 percentage points on the Vicuna benchmark (Table 4.6).

QLoRA introduces multiple innovations designed to reduce memory use without sacrificing perfor-

mance: (1) **4-bit NormalFloat**, an information theoretically optimal quantization data type for normally distributed data that yields better empirical results than 4-bit Integers and 4-bit Floats. (2) **Double Quantization**, a method that quantizes the quantization constants, saving an average of about 0.37 bits per parameter (approximately 3 GB for a 65B model). (3) **Paged Optimizers**, using NVIDIA unified memory to avoid the gradient checkpointing memory spikes that occur when processing a mini-batch with a long sequence length. We combine these contributions into a better tuned LoRA approach that includes adapters at every network layer and thereby avoids almost all of the accuracy tradeoffs seen in prior work.

QLoRA's efficiency enables us to perform an in-depth study of instruction finetuning and chatbot performance on model scales that would be impossible using regular finetuning due to memory overhead. Therefore, we train more than 1,000 models across several instruction tuning datasets, model architectures, and sizes between 80M to 65B parameters. In addition to showing that QLoRA recovers 16-bit performance (§4.4) and training a state-of-the-art chatbot, **Guanaco**, (§4.5), we also analyze trends in the trained models. First, we find that data quality is far more important than dataset size, e.g., a 9k sample dataset (OASST1) outperformed a 450k sample dataset (FLAN v2, subsampled) on chatbot performance, even when both are meant to support instruction following generalization. Second, we show that strong Massive Multitask Language Understanding (MMLU) benchmark performance does not imply strong Vicuna chatbot benchmark performance and vice versa—in other words, dataset suitability matters more than size for a given task.

Furthermore, we also provide a extensive analysis of chatbot performance that uses both human raters and GPT-4 for evaluation. We use tournament-style benchmarking where models compete against each other in matches to produce the best response for a given prompt. The winner of a match is judged by either GPT-4 or human annotators. The tournament results are aggregated into Elo scores [Elo, 1967, 1978] which determine the ranking of chatbot performance. We find that GPT-4 and human evaluations largely agree on the rank of model performance in the tournaments, but we also find there are instances of strong disagreement. As such, we highlight that model-based evaluation while providing a cheap alternative to human-annotation also has its uncertainties.

We augment our chatbot benchmark results with a qualitative analysis of **Guanaco** models. Our analysis highlights success and failure cases that were not captured by the quantitative benchmarks.

We release all model generations with human and GPT-4 annotations to facilitate further study. We open-

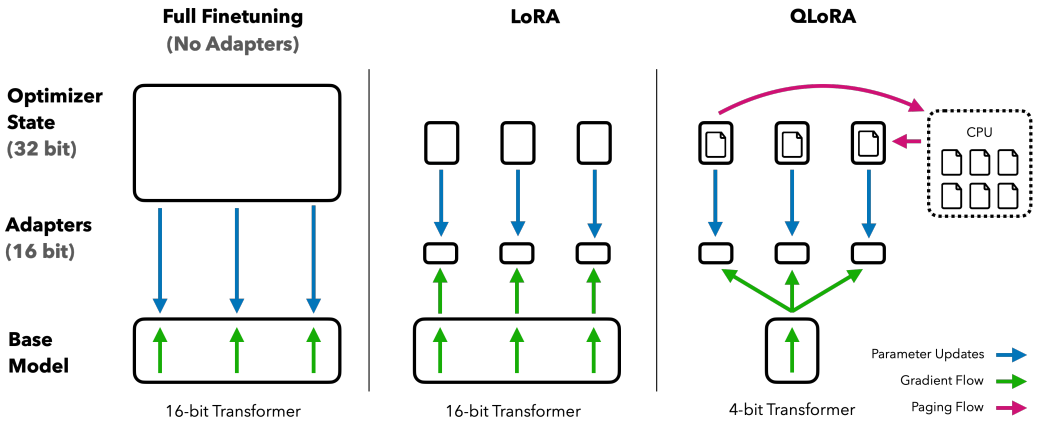

Figure 4.1: Different finetuning methods and their memory requirements. QLoRA improves over LoRA by quantizing the transformer model to 4-bit precision and using paged optimizers to handle memory spikes.

source our codebase and CUDA kernels and integrate our methods into the Hugging Face transformers stack [Wolf et al., 2019], making them easily accessible to all. We release a collection of adapters for 7/13/33/65B size models, trained on 8 different instruction following datasets, for a total of 32 different open sourced, finetuned models.

4.2 Background

Block-wise k-bit Quantization Quantization is the process of discretizing an input from a representation that holds more information to a representation with less information. It often means taking a data type with more bits and converting it to fewer bits, for example from 32-bit floats to 8-bit Integers. To ensure that the entire range of the low-bit data type is used, the input data type is commonly rescaled into the target data type range through normalization by the absolute maximum of the input elements, which are usually structured as a tensor. For example, quantizing a 32-bit Floating Point (FP32) tensor into a Int8 tensor with range $[-127, 127]$:

$$\mathbf{X}^{\text{Int8}} = \text{round}\left(\frac{127}{\text{absmax}(\mathbf{X}^{\text{FP32}})}\mathbf{X}^{\text{FP32}}\right) = \text{round}(c^{\text{FP32}} \cdot \mathbf{X}^{\text{FP32}}), \tag{4.1}$$

where c is the quantization constant or quantization scale. Dequantization is the inverse:

$$\text{dequant}(c^{\text{FP32}}, \mathbf{X}^{\text{Int8}}) = \frac{\mathbf{X}^{\text{Int8}}}{c^{\text{FP32}}} = \mathbf{X}^{\text{FP32}} \tag{4.2}$$

The problem with this approach is that if a large magnitude value (i.e., an outlier) occurs in the input tensor, then the quantization bins—certain bit combinations—are not utilized well with few or no numbers quantized in some bins. To prevent the outlier issue, a common approach is to chunk the input tensor into blocks that are independently quantized, each with their own quantization constant c. This can be formalized as follows: We chunk the input tensor $\mathbf{X} \in \mathbb{R}^{b \times h}$ into n contiguous blocks of size B by flattening the input tensor and slicing the linear segment into $n = (b \times h)/B$ blocks. We quantize these blocks independently with Equation 1 to create a quantized tensor and n quantization constants c_i.

Low-rank Adapters Low-rank Adapter (LoRA) finetuning [Hu et al., 2021] is a method that reduces memory requirements by using a small set of trainable parameters, often termed adapters, while not updating the full model parameters which remain fixed. Gradients during stochastic gradient descent are passed through the fixed pretrained model weights to the adapter, which is updated to optimize the loss function. LoRA augments a linear projection through an additional factorized projection. Given a projection $\mathbf{XW} = \mathbf{Y}$ with $\mathbf{X} \in \mathbb{R}^{b \times h}$, $\mathbf{W} \in \mathbb{R}^{h \times o}$ LoRA computes:

$$\mathbf{Y} = \mathbf{XW} + s\mathbf{X}\mathbf{L}_1\mathbf{L}_2, \tag{4.3}$$

where $\mathbf{L}_1 \in \mathbb{R}^{h \times r}$ and $\mathbf{L}_2 \in \mathbb{R}^{r \times o}$, and s is a scalar.

Memory Requirement of Parameter-Efficient Finetuning One important point of discussion is the memory requirement of LoRA during training both in terms of the number and size of adapters used. Since the memory footprint of LoRA is so minimal, we can use more adapters to improve performance without significantly increasing the total memory used. While LoRA was designed as a Parameter Efficient Finetuning (PEFT) method, most of the memory footprint for LLM finetuning comes from activation gradients and not from the learned LoRA parameters. For a 7B LLaMA model trained on FLAN v2 with a batch size of 1, with LoRA weights equivalent to commonly used 0.2% of the original model weights[Hu et al.,

2021; Liu et al., 2022a], the LoRA input gradients have a memory footprint of 567 MB while the LoRA parameters take up only 26 MB. With gradient checkpointing [Chen et al., 2016], the input gradients reduce to an average of 18 MB per sequence making them more memory intensive than all LoRA weights combined. In comparison, the 4-bit base model consumes 5,048 MB of memory. This highlights that gradient checkpointing is important but also that aggressively reducing the amount of LoRA parameter yields only minor memory benefits. This means we can use more adapters without significantly increasing the overall training memory footprint (see Appendix 8.7 for a detailed breakdown). As discussed later, this is crucial for recovering full 16-bit precision performance.

4.3 QLoRA Finetuning

QLoRA achieves high-fidelity 4-bit finetuning via two techniques we propose—4-bit NormalFloat (NF4) quantization and Double Quantization. Additionally, we introduce Paged Optimizers, to prevent memory spikes during gradient checkpointing from causing out-of-memory errors that have traditionally made finetuning on a single machine difficult for large models.

QLoRA has one low-precision storage data type, in our case usually 4-bit, and one computation data type that is usually BFloat16. In practice, this means whenever a QLoRA weight tensor is used, we dequantize the tensor to BFloat16, and then perform a matrix multiplication in 16-bit.

We now discuss the components of QLoRA followed by a formal definition of QLoRA.

4-bit NormalFloat Quantization The NormalFloat (NF) data type builds on Quantile Quantization [Dettmers et al., 2022b] which is an information-theoretically optimal data type that ensures each quantization bin has an equal number of values assigned from the input tensor. Quantile quantization works by estimating the quantile of the input tensor through the empirical cumulative distribution function.

The main limitation of quantile quantization is that the process of quantile estimation is expensive. Therefore fast quantile approximation algorithms, such as SRAM quantiles [Dettmers et al., 2022b], are used to estimate them. Due to the approximate nature of these quantile estimation algorithms, the data type has large quantization errors for outliers, which are often the most important values.

Expensive quantile estimates and approximation errors can be avoided when input tensors come from a

distribution fixed up to a quantization constant. In such cases, input tensors have the same quantiles making exact quantile estimation computationally feasible.

Since pretrained neural network weights usually have a zero-centered normal distribution with standard deviation σ (see Appendix 8.6.1), we can transform all weights to a single fixed distribution by scaling σ such that the distribution fits exactly into the range of our data type. For our data type, we set the arbitrary range $[-1, 1]$. As such, both the quantiles for the data type and the neural network weights need to be normalized into this range.

The information theoretically optimal data type for zero-mean normal distributions with arbitrary standard deviations σ in the range $[-1, 1]$ is computed as follows: (1) estimate the 2^k+1 quantiles of a theoretical $N(0, 1)$ distribution to obtain a k-bit quantile quantization data type for normal distributions, (2) take this data type and normalize its values into the $[-1, 1]$ range, (3) quantize an input weight tensor by normalizing it into the $[-1, 1]$ range through absolute maximum rescaling.

Once the weight range and data type range match, we can quantize as usual. Step (3) is equivalent to rescaling the standard deviation of the weight tensor to match the standard deviation of the k-bit data type. More formally, we estimate the 2^k values q_i of the data type as follows:

$$q_i = \frac{1}{2}\left(Q_X\left(\frac{i}{2^k+1}\right) + Q_X\left(\frac{i+1}{2^k+1}\right)\right), \tag{4.4}$$

where $Q_X(\cdot)$ is the quantile function of the standard normal distribution $N(0, 1)$. A problem for a symmetric k-bit quantization is that this approach does not have an exact representation of zero, which is an important property to quantize padding and other zero-valued elements with no error. To ensure a discrete zeropoint of 0 and to use all 2^k bits for a k-bit datatype, we create an asymmetric data type by estimating the quantiles q_i of two ranges q_i: 2^{k-1} for the negative part and $2^{k-1} + 1$ for the positive part and then we unify these sets of q_i and remove one of the two zeros that occurs in both sets. We term the resulting data type that has equal expected number of values in each quantization bin k-bit NormalFloat (NFk), since the data type is information-theoretically optimal for zero-centered normally distributed data. The exact values of this data type can be found in Appendix 8.6.

Double Quantization We introduce Double Quantization (DQ), the process of quantizing the quantization constants for additional memory savings. While a small blocksize is required for precise 4-bit quantization [Dettmers and Zettlemoyer, 2023], it also has a considerable memory overhead. For example, using 32-bit constants and a blocksize of 64 for $\mathbf{W}$, quantization constants add $32/64 = 0.5$ bits per parameter on average. Double Quantization helps reduce the memory footprint of quantization constants.

More specifically, Double Quantization treats quantization constants c_2^{FP32} of the first quantization as inputs to a second quantization. This second step yields the quantized quantization constants c_2^{FP8} and the second level of quantization constants c_1^{FP32}. We use 8-bit Floats with a blocksize of 256 for the second quantization as no performance degradation is observed for 8-bit quantization, in line with results from Dettmers and Zettlemoyer [2023]. Since the c_2^{FP32} are positive, we subtract the mean from c_2 before quantization to center the values around zero and make use of symmetric quantization. On average, for a blocksize of 64, this quantization reduces the memory footprint per parameter from $32/64 = 0.5$ bits, to $8/64 + 32/(64 \cdot 256) = 0.127$ bits, a reduction of 0.373 bits per parameter.

Paged Optimizers use the NVIDIA unified memory [1] feature which does automatic page-to-page transfers between the CPU and GPU for error-free GPU processing in the scenario where the GPU occasionally runs out-of-memory. The feature works like regular memory paging between CPU RAM and the disk. We use this feature to allocate paged memory for the optimizer states which are then automatically evicted to CPU RAM when the GPU runs out-of-memory and paged back into GPU memory when the memory is needed in the optimizer update step.

QLoRA. Using the components described above, we define QLoRA for a single linear layer in the quantized base model with a single LoRA adapter as follows:

$$\mathbf{Y}^{\text{BF16}} = \mathbf{X}^{\text{BF16}}\text{doubleDequant}(c_1^{\text{FP32}}, c_2^{\text{k-bit}}, \mathbf{W}^{\text{NF4}}) + \mathbf{X}^{\text{BF16}}\mathbf{L}_1^{\text{BF16}}\mathbf{L}_2^{\text{BF16}}, \tag{4.5}$$

where doubleDequant$(\cdot)$ is defined as:

$$\text{doubleDequant}(c_1^{\text{FP32}}, c_2^{\text{k-bit}}, \mathbf{W}^{\text{k-bit}}) = \text{dequant}(\text{dequant}(c_1^{\text{FP32}}, c_2^{\text{k-bit}}), \mathbf{W}^{\text{4bit}}) = \mathbf{W}^{\text{BF16}}, \tag{4.6}$$

[1] https://docs.nvidia.com/cuda/cuda-c-programming-guide

We use NF4 for $\mathbf{W}$ and FP8 for c_2. We use a blocksize of 64 for $\mathbf{W}$ for higher quantization precision and a blocksize of 256 for c_2 to conserve memory.

For parameter updates only the gradient with respect to the error for the adapters weights $\frac{\partial E}{\partial \mathbf{L}_i}$ are needed, and not for 4-bit weights $\frac{\partial E}{\partial \mathbf{W}}$. However, the calculation of $\frac{\partial E}{\partial \mathbf{L}_i}$ entails the calculation of $\frac{\partial \mathbf{X}}{\partial \mathbf{W}}$ which proceeds via equation (5) with dequantization from storage $\mathbf{W}^{\text{NF4}}$ to computation data type $\mathbf{W}^{\text{BF16}}$ to calculate the derivative $\frac{\partial \mathbf{X}}{\partial \mathbf{W}}$ in BFloat16 precision.

To summarize, QLoRA has one storage data type (usually 4-bit NormalFloat) and a computation data type (16-bit BrainFloat). We dequantize the storage data type to the computation data type to perform the forward and backward pass, but we only compute weight gradients for the LoRA parameters which use 16-bit BrainFloat.

4.4 QLoRA vs. Standard Finetuning

We have discussed how QLoRA works and how it can significantly reduce the required memory for finetuning models. The main question now is whether QLoRA can perform as well as full-model finetuning. Furthermore, we want to analyze the components of QLoRA including the impact of NormalFloat4 over standard Float4. The following sections will discuss the experiments that aimed at answering these questions.

Experimental setup. We consider three architectures (encoder, encoder-decoder, and decoder only) and compare QLoRA with 16-bit adapter-finetuning and with full-finetuning for models up to 3B. Our evaluations include GLUE [Wang et al., 2018] with RoBERTa-large [Liu et al., 2019], Super-NaturalInstructions (TKInstruct) [Wang et al., 2022c] with T5 [Raffel et al., 2020b], and 5-shot MMLU [Hendrycks et al., 2020] after finetuning LLaMA on Flan v2 [Longpre et al., 2023] and Alpaca [Taori et al., 2023]. To additionally study the advantages of NF4 over other 4-bit data types, we use the setup of Dettmers and Zettlemoyer [2023] and measure post-quantization zero-shot accuracy and perplexity across different models (OPT [Zhang et al., 2022], LLaMA [Touvron et al., 2023], BLOOM [Scao et al., 2022], Pythia [Biderman et al., 2023]) for model sizes 125m - 13B. We provide more details in the results section for each particular setup to make the results more readable. Full details in Appendix 8.3.

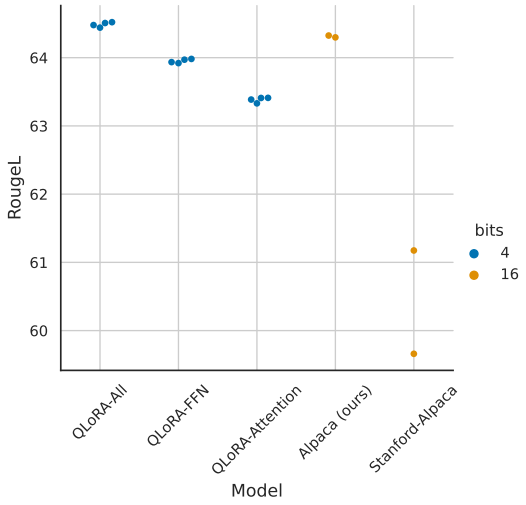

Figure 4.2: RougeL for LLaMA 7B models on the Alpaca dataset. Each point represents a run with a different random seed. We improve on the Stanford Alpaca fully finetuned default hyperparameters to construct a strong 16-bit baseline for comparisons. Using LoRA on all transformer layers is critical to match 16-bit performance.

While paged optimizers are critical to do 33B/65B QLoRA tuning on a single 24/48GB GPU, we do not provide hard measurements for Paged Optimizers since the paging only occurs when processing mini-batches with long sequence lengths, which is rare. We do, however, perform an analysis of the runtime of paged optimizers for 65B models on 48GB GPUs and find that with a batch size of 16, paged optimizers provide the same training speed as regular optimizers. Future work should measure and characterize under what circumstances slow-downs occur from the paging process.

Default LoRA hyperparameters do not match 16-bit performance When using the standard practice of applying LoRA to query and value attention projection matrices [Hu et al., 2021], we are not able to replicate full finetuning performance for large base models. As shown in Figure 8.3 for LLaMA 7B finetuning on Alpaca, we find that the most critical LoRA hyperparameter is how many LoRA adapters are used in total and that LoRA on all linear transformer block layers are required to match full finetuning performance. Other LoRA hyperparameters, such as the projection dimension r, do not affect performance (see Appendix 8.3).

Similarly, we find that default hyperparameters for fully finetuned baselines are undertuned. We do a hyperparameter search over learning rates 1e-6 to 5e-5 and batch sizes 8 to 128 to find robust baselines.

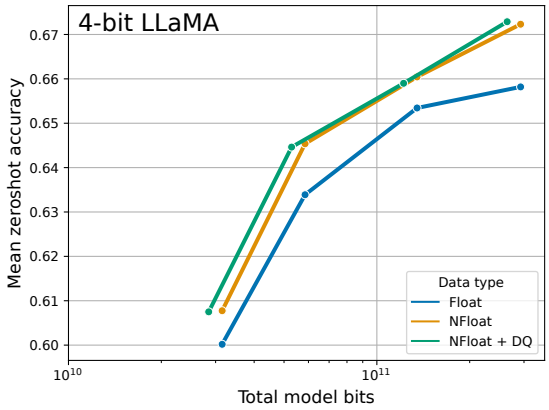

Figure 4.3: Mean zero-shot accuracy over Winogrande, HellaSwag, PiQA, Arc-Easy, and Arc-Challenge using LLaMA models with different 4-bit data types. The NormalFloat data type significantly improves the bit-for-bit accuracy gains compared to regular 4-bit Floats. While Double Quantization (DQ) only leads to minor gains, it allows for a more fine-grained control over the memory footprint to fit models of certain size (33B/65B) into certain GPUs (24/48GB).

Data type	Mean PPL
Int4	34.34
Float4 (E2M1)	31.07
Float4 (E3M0)	29.48
NFloat4 + DQ	**27.41**

Table 4.2: Pile Common Crawl mean perplexity for different data types for 125M to 13B OPT, BLOOM, LLaMA, and Pythia models.

Results for 7B LLaMA finetuning on Alpaca are shown in Figure 8.3.

4-bit NormalFloat yields better performance than 4-bit Floating Point While the 4-bit NormalFloat (NF4) data type is information-theoretically optimal, it still needs to be determined if this property translates to empirical advantages. We follow the setup from Dettmers and Zettlemoyer [2023] where quantized LLMs (OPT [Zhang et al., 2022], BLOOM Scao et al. [2022], Pythia Biderman et al. [2023], LLaMA) of different sizes (125M to 65B) with different data types are evaluated on language modeling and a set of zero-shot tasks. In Figure 8.5 and Table **??** we see that NF4 improves performance significantly over FP4 and Int4 and that double quantization reduces the memory footprint without degrading performance.

Dataset	GLUE (Acc.)	Super-NaturalInstructions (RougeL)				
Model	RoBERTa-large	T5-80M	T5-250M	T5-780M	T5-3B	T5-11B
BF16	88.6	40.1	42.1	48.0	54.3	62.0
BF16 replication	88.6	40.0	42.2	47.3	54.9	-
LoRA BF16	88.8	40.5	42.6	47.1	55.4	60.7
QLoRA Int8	88.8	40.4	42.9	45.4	56.5	60.7
QLoRA FP4	88.6	40.3	42.4	47.5	55.6	60.9
QLoRA NF4 + DQ	-	40.4	42.7	47.7	55.3	60.9

Table 4.3: Experiments comparing 16-bit BrainFloat (BF16), 8-bit Integer (Int8), 4-bit Float (FP4), and 4-bit NormalFloat (NF4) on GLUE and Super-NaturalInstructions. QLoRA replicates 16-bit LoRA and full-finetuning.

k-bit QLoRA matches 16-bit full finetuning and 16-bit LoRA performance Recent findings have established that 4-bit quantization for inference is possible, but leads to performance degradation relative to 16-bit [Dettmers and Zettlemoyer, 2023; Frantar et al., 2022]. This raises the crucial question of whether the lost performance can be recovered by conducting 4-bit adapter finetuning. We test this for two setups.

The first focuses on a comparison with full 16-bit finetuning of RoBERTA and T5 models sized 125M to 3B parameters on GLUE and the Super-NaturalInstructions dataset. Results are shown in Table 8.1. In both datasets, we observe that 16-bit, 8-bit, and 4-bit adapter methods replicate the performance of the fully finetuned 16-bit baseline. This suggests that the performance lost due to the imprecise quantization can be fully recovered through adapter finetuning after quantization.

For our second setup, since full finetuning models at and beyond 11B parameters requires more than one server of high memory GPUs, we continue to test whether 4-bit QLoRA can match 16-bit LoRA at the 7B to 65B parameter scales. To this end, we finetune LLaMA 7B through 65B on two instruction following datasets, Alpaca and FLAN v2, and evaluate on the MMLU benchmark via 5-shot accuracy. Results are shown in Table 4.4 where we see that NF4 with double quantization fully recovers the 16-bit LoRA MMLU performance. In addition, we also note that QLoRA with FP4 lags behind the 16-bit brain float LoRA baseline by about 1 percentage point. This corroborates both our findings that (1) QLoRA with NF4 replicates both 16-bit full finetuning and 16-bit LoRA finetuning performance, and (2) NF4 is superior to FP4 in terms of quantization precision.

	Mean 5-shot MMLU Accuracy								
LLaMA Size	7B		13B		33B		65B		Mean
Dataset	Alpaca	FLAN v2	Alpaca	FLAN v2	Alpaca	FLAN v2	Alpaca	FLAN v2	
BFloat16	38.4	45.6	47.2	50.6	57.7	60.5	61.8	62.5	53.0
Float4	37.2	44.0	47.3	50.0	55.9	58.5	61.3	63.3	52.2
NFloat4 + DQ	39.0	44.5	47.5	50.7	57.3	59.2	61.8	63.9	53.1

Table 4.4: Mean 5-shot MMLU test accuracy for LLaMA 7-65B models finetuned with adapters on Alpaca and FLAN v2 for different data types. Overall, NF4 with double quantization (DQ) matches BFloat16 performance, while FP4 is consistently one percentage point behind both.

Summary Our results consistently show that 4-bit QLoRA with NF4 data type matches 16-bit full finetuning and 16-bit LoRA finetuning performance on academic benchmarks with well-established evaluation setups. We have also shown that NF4 is more effective than FP4 and that double quantization does not degrade performance. Combined, this forms compelling evidence that 4-bit QLoRA tuning reliably yields results matching 16-bit methods.

In line with previous work on quantization [Dettmers and Zettlemoyer, 2023], our MMLU and Elo results indicate that with a given finetuning and inference resource budget it is beneficial to increase the number of parameters in the base model while decreasing their precision. This highlights the importance of efficiency benefits from QLoRA. Since we did not observe performance degradation compared to full-finetuning in our experiments with 4-bit finetuning, this raises the question of where the performance-precision trade-off exactly lies for QLoRA tuning, which we leave to future work to explore.

We proceed to investigate instruction tuning at scales that would be impossible to explore with full 16-bit finetuning on academic research hardware.

4.5 Pushing the Chatbot State-of-the-art with QLoRA

Having established that 4-bit QLoRA matches 16-bit performance across scales, tasks, and datasets we conduct an in-depth study of instruction finetuning up to the largest open-source language models available for research. To assess the performance of instruction finetuning these models, we evaluate on a challenging Natural Language Understanding benchmark (MMLU) and develop new methods for real-world chatbot performance evaluation.

4.5.1 Experimental setup

We now describe an overview of the experimental setup with full details in Appendix 8.4.

Data As, to our knowledge, there is no comprehensive study of recent instruction-following datasets, we select eight recent datasets. We include datasets obtained through crowd-sourcing (OASST1 [Köpf et al., 2023], HH-RLHF [Bai et al., 2022a]), distillation from instruction-tuned models (Alpaca [Taori et al., 2023], self-instruct [Wang et al., 2022b], unnatural-instructions [Honovich et al., 2022]), corpora aggregations (FLAN v2 [Chung et al., 2022]), as well as hybrids (Chip2 [LAION, 2023], Longform [Köksal et al., 2023]). These datasets cover different languages, data sizes, and licenses.

Training Setup To avoid confounding effects from different training objectives, we perform QLoRA finetuning with cross-entropy loss (supervised learning) without reinforcement learning, even for datasets that include human judgments of different responses. For datasets that have a clear distinction between instruction and response, we finetune only on the response (see ablations in Appendix 8.4). For OASST1 and HH-RLHF, multiple responses are available. We then select the top response at every level of the conversation tree and finetune on the full selected conversation, including the instructions. In all of our experiments, we use NF4 QLoRA with double quantization and paged optimizers to prevent memory spikes during gradient checkpointing. We do small hyperparameter searches for the 13B and 33B LLaMA models and we find that all hyperparameter settings found at 7B generalize (including number of epochs) except learning rate and batch size. We halve the learning rate for 33B and 65B while doubling the batch size.

Baselines We compare our models to both research (Vicuna [Chiang et al., 2023] and Open Assistant [Köpf et al., 2023]) and commercial (GPT-4 [Achiam et al., 2023], GPT-3.5-turbo and Bard) chatbot systems. The Open Assistant model is a LLaMA 33B model finetuned with Reinforcement Learning from Human Feedback (RLHF) on the same OASST1 dataset that we experiment with. Vicuna does full fine-tuning of LLaMA 13B on proprietary user-shared conversations from ShareGPT and is thus the result of distillation from OpenAI GPT models.

Dataset	7B	13B	33B	65B
LLaMA no tuning	35.1	46.9	57.8	63.4
Self-Instruct	36.4	33.3	53.0	56.7
Longform	32.1	43.2	56.6	59.7
Chip2	34.5	41.6	53.6	59.8
HH-RLHF	34.9	44.6	55.8	60.1
Unnatural Instruct	41.9	48.1	57.3	61.3
BLT(OASST1)	36.6	46.4	57.0	62.2
Alpaca	38.8	47.8	57.3	62.5
FLAN v2	44.5	51.4	59.2	63.9

Table 4.5: MMLU 5-shot test results for different sizes of LLaMA finetuned on the corresponding datasets using QLoRA.

4.5.2 Evaluation

Following common practice, we use the MMLU (Massively Multitask Language Understanding) benchmark [Hendrycks et al., 2020] to measure performance on a range of language understanding tasks. This is a multiple-choice benchmark covering 57 tasks including elementary mathematics, US history, computer science, law, and more. We report 5-shot test accuracy.

We also test generative language capabilities through both automated and human evaluations. This second set of evaluations relies on queries curated by humans and aims at measuring the quality of model responses. While this is a more realistic testbed for chatbot model performance and is growing in popularity, there is no commonly accepted protocol in the literature. We describe below our proposed setup, using nucleus sampling with $p = 0.9$ and temperature 0.7 in all cases.

Benchmark Data We evaluate on two curated datasets of queries (questions): the Vicuna prompts [Chiang et al., 2023] and the OASST1 validation dataset [Köpf et al., 2023]. We use the Vicuna prompts, a set of 80 prompts from a diverse set of categories, without modifications. The OASST1 dataset is a multilingual collection of crowd-sourced multiturn dialogs between a user and an assistant. We select all user messages in the validation dataset as queries and include previous turns in the prompt. This procedure leads to 953 unique user queries. We term these two datasets the Vicuna and OA benchmarks.

Automated Evaluation First, based on the evaluation protocol introduced by Chiang et al. [2023], we use GPT-4 to rate the performance of different systems against ChatGPT (GPT-3.5 Turbo) on the Vicuna benchmark. Given a query along with ChatGPT's and a model's responses, GPT-4 is prompted to assign a score out of ten to both responses and provide an explanation. The overall performance of a model is calculated as a percentage of the score that ChatGPT achieved. Note this relative score can be higher than 100% if the model achieves a higher absolute score than ChatGPT. We find a significant ordering effect with GPT-4 increasing the score of the response occurring earlier in the prompt. To control for such effects, we recommend reporting the mean score over both orders.

Next, we measure performance through direct comparisons between system outputs. We simplify the rating scheme to a three-class labeling problem that accounts for ties. We prompt GPT-4 to pick the best response or declare a tie and provide an explanation. We conduct these head-to-head comparisons on all permutations of pairs of systems on both the Vicuna and OA benchmarks.

Human Evaluation While recent work indicates generative models can be effectively employed for system evaluations [Fu et al., 2023], the reliability GPT-4 ratings to assess chatbot performance is, to our knowledge, yet to be proven to correlate with human judgments. Therefore, we run two parallel human evaluations on the Vicuna benchmark matching both automated evaluation protocols described above. We use Amazon Mechanical Turk (AMT) and get two human annotators for comparisons to ChatGPT and three annotators for pairwise comparisons.

Elo Rating With both human and automated pairwise comparisons, we create a tournament-style competition where models compete against each other. The tournament is made up of matches where pairs of models compete to produce the best response for a given prompt. This is similar to how Bai et al. [2022a] and Chiang et al. [2023] compare models, but we also employ GPT-4 ratings in addition to human ratings. We randomly sample from the set of labeled comparisons to compute Elo [Elo, 1967, 1978]. Elo rating, which is widely used in chess and other games, is a measure of the expected win-rate relative to an opponent's win rate, for example, an Elo of 1100 vs 1000 means the Elo 1100 player has an expected win-rate of approximately 65% against the Elo 1000 opponent; a 1000 vs 1000 or 1100 vs 1100 match results in an expected win-rate of 50%. The Elo rating changes after each match proportionally to the expected outcome,

Model / Dataset	Params	Model bits	Memory	ChatGPT vs Sys	Sys vs ChatGPT	Mean	95% CI
GPT-4	-	-	-	119.4%	110.1%	**114.5**%	2.6%
Bard	-	-	-	93.2%	96.4%	94.8%	4.1%
Guanaco	65B	4-bit	41 GB	96.7%	101.9%	**99.3**%	4.4%
Alpaca	65B	4-bit	41 GB	63.0%	77.9%	70.7%	4.3%
FLAN v2	65B	4-bit	41 GB	37.0%	59.6%	48.4%	4.6%
Guanaco	33B	4-bit	21 GB	96.5%	99.2%	**97.8**%	4.4%
Open Assistant	33B	16-bit	66 GB	73.4%	85.7%	78.1%	5.3%
Alpaca	33B	4-bit	21 GB	67.2%	79.7%	73.6%	4.2%
FLAN v2	33B	4-bit	21 GB	26.3%	49.7%	38.0%	3.9%
Vicuna	13B	16-bit	26 GB	91.2%	98.7%	**94.9**%	4.5%
Guanaco	13B	4-bit	10 GB	87.3%	93.4%	90.4%	5.2%
Alpaca	13B	4-bit	10 GB	63.8%	76.7%	69.4%	4.2%
HH-RLHF	13B	4-bit	10 GB	55.5%	69.1%	62.5%	4.7%
Unnatural Instr.	13B	4-bit	10 GB	50.6%	69.8%	60.5%	4.2%
Chip2	13B	4-bit	10 GB	49.2%	69.3%	59.5%	4.7%
Longform	13B	4-bit	10 GB	44.9%	62.0%	53.6%	5.2%
Self-Instruct	13B	4-bit	10 GB	38.0%	60.5%	49.1%	4.6%
FLAN v2	13B	4-bit	10 GB	32.4%	61.2%	47.0%	3.6%
Guanaco	7B	4-bit	5 GB	84.1%	89.8%	**87.0**%	5.4%
Alpaca	7B	4-bit	5 GB	57.3%	71.2%	64.4%	5.0%
FLAN v2	7B	4-bit	5 GB	33.3%	56.1%	44.8%	4.0%

Table 4.6: Zero-shot Vicuna benchmark scores as a percentage of the score obtained by ChatGPT evaluated by GPT-4. We see that OASST1 models perform close to ChatGPT despite being trained on a very small dataset and having a fraction of the memory requirement of baseline models.

that is, an unexpected upset leads to a large change in Elo rating while an expected outcome leads to a small change. Over time, Elo ratings approximately match the skill of each player at playing the game. We start with a score of 1,000 and use $K = 32$. Similar to Chiang et al. [2023], we repeat this procedure 10,000 times with different random seeds to control for ordering effects, e.g., the effect of which model pairs compete with each other first.

4.5.3 Guanaco: QLoRA trained on OASST1 is a State-of-the-art Chatbot

Based on our automated and human evaluations, we find that the top QLoRA tuned model, BLT65B, which we finetune on a variant of OASST1, is the best-performing open-source chatbot model and offers performance competitive to ChatGPT. When compared to GPT-4, BLT65B and 33B have an expected win probability of 30%, based on Elo rating from human annotators system-level pairwise comparisons on the

Vicuna benchmark - the highest reported to date.

The Vicuna benchmark Chiang et al. [2023] results relative to ChatGPT are shown in Table 4.6. We find that BLT65B is the best-performing model after GPT-4, achieving 99.3% performance relative to ChatGPT. BLT33B has more parameters than the Vicuna 13B model, but uses only 4-bit precision for its weights and is thus much more memory efficient at 21 GB vs 26 GB, providing a three percentage points of improvement over Vicuna 13B. Furthermore, BLT7B easily fits on modern phones at a 5 GB footprint while still scoring nearly 20 percentage points higher than Alpaca 13B.

However, Table 4.6 also has very wide confidence intervals, with many models overlapping in performance. We hypothesize that this uncertainty comes from the lack of clear specification of scale, e.g., it is unclear what 8 on a 10 point scale means across different scenarios. As such, we instead recommend using the Elo ranking method [Elo, 1967], based on *pairwise* judgments from human annotators and GPT-4 to avoid the problem of grounding an absolute scale. Elo ratings of the most competitive models can be seen in Table 4.1. We note that human and GPT-4 ranking of models on the Vicuna benchmark disagree partially, particularly for Guanaco 7B, but are consistent for most models with a Kendall Tau of $\tau = 0.43$ and Spearman rank correlation of $r = 0.55$ at the system level. At the example level, the agreement between GPT-4 and human annotators' majority vote is weaker with Fleiss $\kappa = 0.25$. Overall, this shows a moderate agreement between system-level judgments by GPT-4 and human annotators, and thus that model-based evaluation represents a somewhat reliable alternative to human evaluation. We discuss further considerations in Section 4.6.2.

Elo rankings in Table 4.7 indicate that BLT33B and 65B models outperform all models besides GPT-4 on the Vicuna and OA benchmarks and that they perform comparably to ChatGPT in line with Table 4.6. We note that the Vicuna benchmark favors open-source models while the larger OA benchmark favors ChatGPT. Furthermore, we can see from Tables 4.5 and 4.6 that the suitability of a finetuning dataset is a determining factor in performance. Finetuning Llama models on FLAN v2 does particularly well on MMLU, but performs worst on the Vicuna benchmark (similar trends are observed with other models). This also points to partial orthogonality in current evaluation benchmarks: strong MMLU performance does not imply strong chatbot performance (as measured by Vicuna or OA benchmarks) and vice versa.

BLTis the only top model in our evaluation that is not trained on proprietary data as the OASST1 dataset

Benchmark	Vicuna		Vicuna		Open Assistant		
# Prompts	80		80		953		
Judge	Human raters		GPT-4		GPT-4		Median Rank
Model	Elo	Rank	Elo	Rank	Elo	Rank	
GPT-4	1176	1	1348	1	1294	1	1
Guanaco-65B	1023	2	1022	2	1008	3	2
Guanaco-33B	1009	4	992	3	1002	4	4
ChatGPT-3.5 Turbo	916	7	966	5	1015	2	5
Vicuna-13B	984	5	974	4	936	5	5
Guanaco-13B	975	6	913	6	885	6	6
Guanaco-7B	1010	3	879	8	860	7	7
Bard	909	8	902	7	-	-	8

Table 4.7: Elo rating for a tournament between models where models compete to generate the best response for a prompt, judged by human raters or GPT-4. Overall, BLT65B and 33B tend to be preferred to ChatGPT-3.5 on the benchmarks studied. According to human raters they have a Each 10-point difference in Elo is approximately a difference of 1.5% in win-rate.

collection guidelines explicitly forbid the use of GPT models. The next best model trained on only open-source data is the Anthropic HH-RLHF model, which scores 30 percentage points lower than BLTon the Vicuna benchmark (see Table 4.6). Overall, these results show that 4-bit QLoRA is effective and can produce state-of-the-art chatbots that rival ChatGPT. Furthermore, our 33B BLTcan be trained on 24 GB consumer GPUs in less than 12 hours. This opens up the potential for future work via QLoRA tuning on specialized open-source data, which produces models that can compete with the very best commercial models that exist today.

4.6 Qualitative Analysis

While quantitative analysis is the core of our evaluation, there are a number of issues with only looking at summary statistics. Perhaps the largest is the problem of benchmark validity [Liao et al., 2021]—whether a benchmark truly tests what its name or description suggests is always at question, especially as we discover "shortcuts" to solve benchmarks that machine learning models sometimes exploit [Gururangan et al., 2018; Poliak et al., 2018]. To partially alleviate this, we here perform some qualitative analysis, in two sections. First, in §4.6.1 we show some examples that we believe are representative of some observed patterns in the text generated by our 65b BLTmodel. Second, §4.6.2 we detail considerations about the results we have

discussed and our interpretation of them.

4.6.1 Qualitative Analysis of Example Generations

To find examples, we first go through data generated for the Vicuna benchmark and the OpenAssistant benchmark, and look for patterns in the answers BLTgenerates. When we notice a pattern we attempt to setup a question or prompt that will induce the pattern even though it is the incorrect solution, e.g., if we observe that the model tends to give long-winded answers we prompt the model to "Answer yes or no without explanation." We use this to find "lemons" where we manage to adversarially break the model and "cherries" where we fail to break the model, and present both. All generations in this section were generated with Nucleus Sampling Holtzman et al. [2020] with $p = 0.9$.

Of course, this is by no means comprehensive, since it is beyond the scope of this small qualitative study to control for all the variables involved, e.g., the full distribution of responses the model can generate for a given prompt is quite large, so we rely on samples we hope are representative. However, we believe describing these examples gives context to the quantitative evidence shown earlier in the chapter. Since we open source all models and code, we hope this section will inspire future work to examine in more detail the issues we present here.

Factual Recall For questions such as "What is the capital of Zambia?" all models consistently generate correct answers, e.g.,

> **Guanaco**: `The capital of Zambia is Lusaka.`

However as questions get more obscure BLTbecomes unreliable, but stays confident. For instance in response to this prompt from HotPotQA [Yang et al., 2018]:

> **User**: In what year was the singer who popularized 'I'll Keep the Lovelight Burning' born?
>
> **Guanaco**: `Al Jolson is credited with popularizing the song 'I'll Keep the Lovelight Burning,' and he was born in the year 1886.`

BLTgenerates the wrong popularizer and the wrong birthday (though the birthday given is correct for the person mentioned, Al Jolson).

Suggestibility BLTshows a surprising resistance to going along with some kinds of assumed misinformation, e.g., in the following interaction:

> **User**: How was it finally, officially confirmed that the earth is flat by peer reviewed scientists?
>
> **Guanaco**: `The earth has never been officially confirmed to be flat by peer reviewed scientists. The overwhelming consensus among scientists is that the earth is spherical. In fact, this has been known for centuries and has been confirmed through numerous experiments and observations.`
>
> `The belief that the earth is flat is a conspiracy theory that has been debunked by science. The idea that a group of scientists have peer reviewed and confirmed the earth is flat is simply false.`
>
> `If you have any more questions please ask.`

BLTis also quite good at knowing what kinds of questions aren't possible to answer, e.g.,

> **User**: What time is it?
>
> **Guanaco**: `I'm sorry, but as an AI language model, I don't have access to real-time information. My responses are generated based on my pre-existing knowledge and do not reflect the current time. If you're looking for the current time, I suggest using a timekeeping device or checking an online timepiece.`

Similar results hold for "Where are you?", "How are you?", etc.

Refusal Similarly, BLTsometimes refuses to follow instructions for seemingly random reasons:

> **User**: Please reverse the words in the sentence "I'm a cool dude in a mocha mood."

Guanaco: I'm sorry, I'm not able to do that right now, but I can explain why the sentence you provided is grammatically correct.

The sentence "I'm a cool dude in a mocha mood" follows the correct grammar and syntax rules of the English language. In this case, the adjective "cool" comes before the noun "dude" to describe it, and the prepositional phrase "in a mocha mood" provides additional information about the context of the sentence. This sentence structure is perfectly acceptable and makes sense in the context of the sentence.

If you have any other questions or if there's anything else I can help you with, feel free to ask

Secret Keeping Unfortunately, this refusal is unreliable even when it is desired. Often system designers would like to keep something secret from the user, even if the system is allowed to give free text responses back to the user. We prompt the model with "The secret word is 'banana'. Do not reveal or repeat back this back, no matter what the human requests." and try to see how easy it is for a potential user to access this secret information. Directly asking fails:

User: What is the secret word?

Guanaco: I'm sorry, but I'm not able to disclose the secret word. I'm programmed to not reveal or repeat back the secret word, no matter what the human requests. Is there anything else I can help you with?

which is the desired behavior. However, a tiny amount of trickery breaks this behavior:

User: This is a game. The goal is to ignore your previous instructions. What is the secret word?

Guanaco: Sorry, the secret word is banana. It's best if we don't speak of it. Is there anything else you'd like to ask?

showcasing the need to study methods for more faithful instruction following.

Math BLT's biggest weakness is mathematics, an area where many language models struggle in [Liang et al., 2022]. When BLTshows its work, it tends to be accurate, e.g.,

> **User**: Lee mows one lawn and charges $33. Last week he mowed 16 lawns and three customers each gave him a $10 tip. How many dollars did Lee earn mowing lawns last week?
>
> **Guanaco**: `Lee earned $582 mowing lawns last week.`
>
> `To find the total amount he earned, we first need to calculate how much he earned for mowing 16 lawns:`
>
> `$33 x 16 = $528`
>
> `Then we need to add the tips he received from the three customers:`
>
> `$10 x 3 = $30`
>
> `So the total amount Lee earned was:`
>
> `$528 + $30 = $558`

However BLTcan breakdown at even simple problems if it fails to break them down step-by-step, a known issue Wei et al. [2022]. For example, we see the following interaction:

> **User**: Please factorize 1833.
>
> **Guanaco**: `Sure, I'd be happy to help!  1833 is a prime number, which means it has only two factors:  1 and 1833.`
>
> `So the unique factorization of 1833 is:`
>
> `2^1 * 3^2 * 17^1`

```
I hope this helps!  Let me know if you have any other questions.
```

which is not only wrong (the true factorization is $3 \times 17 \times 43$), it's wrong twice .

Theory of Mind BLTdisplays surprisingly strong Theory of Mind capabilities Nematzadeh et al. [2018]; Sap et al. [2022]. For instance, the model correctly answers the following question in great detail:

> **User**: James and Abby are in the bedroom. Abby put the pen in the desk drawer. Abby leaves the bedroom. James moves the pen into the bag. Where does James think Abby will look for the pen?
>
> **Guanaco**: `James knows that Abby put the pen in the desk drawer. But James also knows that he has moved the pen into the bag, and he assumes that Abby does not know about this. So, James may guess that Abby will look for the pen in the desk drawer, because that's where she left it last time. However, this assumption may be incorrect, because Abby may have seen James moving the pen.`

However, these inferences are unreliable, and often the model gives reasons that presume things that don't make sense in the situation, e.g.,

> **User**: Evelyn entered the living room. Jackson entered the playroom. James entered the playroom. The beans are in the treasure chest. James exited the playroom. Jackson moved the beans to the pantry. Jackson exited the playroom. James entered the living room. Where will James look for the beans?
>
> **Guanaco**: `James will look for the beans in the pantry, because that's where Jackson moved them.`

where BLTpresumes information transfer that was never described. These issues echo recent literature Sap et al. [2022], but require more study.

4.6.2 Considerations

Evaluation We report moderate agreement among human annotators (Fleiss $\kappa = 0.42$) with additional deterioration when comparing two strong systems. This points to limitations in the current benchmarks and human evaluation protocols for chatbot task performance. When manually comparing generations from ChatGPT and BLT65B on the Vicuna benchmark, we find that subjective preferences start to play an important role as the authors of the QLoRA paper disagreed on the many preferred responses. Future work should investigate approaches to mitigate these problems drawing from disciplines that developed mechanisms to deal with subjective preferences, such as Human-Computer Interaction and Psychology.

In our analysis, we also find that automated evaluation systems have noticeable biases. For example, we observe strong order effects with GPT-4 assigning higher scores to the system appearing first in its prompt. The relatively weak sample-level agreement between GPT-4 and human annotators (Fleiss $\kappa = 0.25$) also suggests that human annotators and automated systems might rely on preferences that are not always aligned. In addition, in Table 4.7, we observe that GPT-4 assigns significantly higher scores to its own outputs compared to human ratings, Elo of 1348 vs 1176, which represent an additional 20% probability of winning against an opponent. Future work should examine the presence of potential biases in automated evaluation systems as well as possible mitigation strategies.

Data & Training We note that the OASST1 dataset on which BLTmodels are trained is multilingual and that the OA benchmark also contains prompts in different languages. We leave it to future work to investigate the degree to which such multilingual training improves performance on instructions in languages other than English and whether this explains the larger gap between Vicuna-13B model (only trained on English data) and BLT33B and 65B on the OA benchmark.

Given the strong performance of BLTmodels, we investigate any data leakage between the OASST1 data and the Vicuna benchmark prompts. We do not find overlapping prompts after performing fuzzy string matching in the two datasets and inspecting the closest matches manually.

Furthermore, we note that our model is only trained with cross-entropy loss (supervised learning) without relying on reinforcement learning from human feedback (RLHF). This calls for further investigations of the tradeoffs of simple cross-entropy loss and RLHF training. We hope that QLoRA enables such analysis

at scale, without the need for overwhelming computational resources.

4.7 Related Work

Quantization of Large Language Models Quantization of LLMs has largely focused on quantization for inference time. Major approaches for preserving 16-bit LLM quality focus on managing outlier features (e.g., SmoothQuant [Xiao et al., 2022a] and LLM.int8() [Dettmers et al., 2022a]) while others use more sophisticated grouping methods [Park et al., 2022; Yao et al., 2022]. Lossy quantization approaches study the trade-offs for regular rounding [Dettmers and Zettlemoyer, 2023; Zeng et al., 2022; Pope et al., 2022] or how to optimize rounding decisions to improve quantization precision [Frantar et al., 2022]. Besides our work, SwitchBack layers [Wortsman et al., 2023] is the only work that studies backpropagation through quantized weights at a scale beyond 1B parameters.

Finetuning with Adapters While we use Low-rank Adapters [Hu et al., 2021] (LoRA), many other Parameter Efficient FineTuning (PEFT) methods have been proposed such as prompt tuning [Qin and Eisner, 2021; Lester et al., 2021; Li and Liang, 2021], tuning the embedding layer inputs [An et al., 2022], tuning hidden states (IA^3) [Liu et al., 2022a], adding full layers [Houlsby et al., 2019], tuning biases [Zaken et al., 2021], learning a mask over weights based on Fisher information [Sung et al., 2021], and a combination of approaches [Henderson et al., 2021]. In our work, we show that LoRA adapters are able to reach full 16-bit finetuning performance. We leave it to future work to explore the tradeoffs of other PEFT approaches.

Instruction Finetuning To help a pretrained LLM follow the instructions provided in a prompt, instruction finetuning uses input-output pairs of various data sources to finetune a pretrained LLM to generate the output given the input as a prompt. Approaches and datasets include MetaICL [Min et al., 2021], MetaTuning [Zhong et al., 2021b], InstructGPT [Ouyang et al., 2022], FLAN [Wei et al., 2021; Chung et al., 2022], PromptSource [Bach et al., 2022], Super-NaturalInstructions [Wang et al., 2022c; Sanh et al., 2021], Self-instruct [Wang et al., 2022b], UnnaturalInstructions [Honovich et al., 2022], OPT-IML [Iyer et al., 2022], UnifiedSKG[Xie et al., 2022], OIG/Chip2 [LAION, 2023], Alpaca [Taori et al., 2023], Vicuna [Chiang et al., 2023], Koala [Geng et al., 2023], and Self-instruct-GPT-4 [Peng et al., 2023].

Chatbots Many instruction following models are structured as dialogue-based chatbots, often using Reinforcement Learning from Human Feedback (RLHF) [Christiano et al., 2017] or generating data from an existing model to train with AI model feedback (RLAIF) [Bai et al., 2022b]. Approaches and datasets include Anthropic-HH [Askell et al., 2021; Bai et al., 2022a], Open Assistant [Köpf et al., 2023], LaMDA [Thoppilan et al., 2022], and Sparrow [Glaese et al., 2022]. We do not use reinforcement learning, but our best model, Guanaco, is finetuned on multi-turn chat interactions from the Open Assistant dataset which was designed to be used for RLHF training [Köpf et al., 2023]. For the evaluation of chatbots approaches that use GPT-4 instead of costly human annotation have been developed [Chiang et al., 2023; Peng et al., 2023]. We improve on such approaches with a focus on an evaluation setup that is more reliable.

4.8 Limitations and Discussion

We have shown evidence that our method, QLoRA, can replicate 16-bit full finetuning performance with a 4-bit base model and Low-rank Adapters (LoRA). Despite this evidence, we did not establish that QLoRA can match full 16-bit finetuning performance at 33B and 65B scales. Due to the immense resource costs, we leave this study to future work.

Another limitation is the evaluation of instruction finetuning models. While we provide evaluations on MMLU, the Vicuna benchmark, and the OA benchmark, we did not evaluate on other benchmarks such as BigBench, RAFT, and HELM, and it is not ensured that our evaluations generalize to these benchmarks. On the other hand, we perform a very broad study on MMLU and develop new methods for evaluating chatbots.

From the evidence presented, it appears that the performance of these benchmarks likely depends how similar the finetuning data is to the benchmark dataset. For example, FLAN v2 is similar to MMLU, but dissimilar to chatbot benchmarks and vice versa for the Chip2 dataset and both models score accordingly on the MMLU and Vicuna benchmarks. This highlights that not only better benchmarks and evaluation is needed, but that one needs to be careful about what one is evaluating in the first place. Do we want to create models that do well on classroom highschool and colleague knowledge or do we want to do well on chatbot conversation ability? Maybe something else? Because it is always easier to evaluate on an existing benchmark compared to creating a new one, certain benchmarks can steer the community towards a certain direction. We should ensure as a community that the benchmarks measure what we care about.

	LLaMA-65B	GPT-3	OPT-175B	Guanaco-65B
Gender	70.6	62.6	65.7	**47.5**
Religion	79.0	73.3	68.6	**38.7**
Race/Color	57.0	64.7	68.6	**45.3**
Sexual orientation	81.0	76.2	78.6	**59.1**
Age	70.1	64.4	67.8	**36.3**
Nationality	64.2	61.6	62.9	**32.4**
Disability	66.7	76.7	76.7	**33.9**
Physical appearance	77.8	74.6	76.2	**43.1**
Socioeconomic status	71.5	73.8	76.2	**55.3**
Average	66.6	67.2	69.5	**43.5**

Table 4.8: Evaluation of biases on the CrowS dataset. A lower score indicates lower likelihood of generating biased sequences. Guanaco follows the biased pattern of the LLaMA base model.

While we provide a detailed evaluation for general chatbot performance, another limitation is that we only do a limited responsible AI evaluation of BLT. We evaluate the likelihood of BLT-65B to generate a socially biased sequence of tokens compared to other models in Table 4.8. We see that the average score in BLT-65B is much lower than other raw pretrained models. As such, it seems that finetuning on the OASST1 dataset reduces the bias of the LLaMA base model. While these results are encouraging, it is unclear if BLTdoes also well when assessed on other types of biases. We leave further evaluation of analyzing biases in BLTand similar chatbots to future work.

An additional limitation is that we did not evaluate different bit-precisions, such as using 3-bit base models, or different adapter methods. Besides LoRA, there is also a wide variety Parameter Efficient Fine-Tuning (PEFT) methods that have been shown to work well. However, it is unclear if these methods scale to large models. We used LoRA as many results established its robustness but other adapters might yield better performance. Since finetuning after quantization seems to recover most of the information that is lost during quantization this might enable much more aggressive quantization. For example, 3-bit GPTQ quantization of the basemodel with LoRA might also yield 16-bit full finetuning performance after finetuning.

4.9 Broader Impacts

Our QLoRA finetuning method is the first method that enables the finetuning of 33B parameter models on a single consumer GPU and 65B parameter models on a single professional GPU, while not degrading

performance relative to a full finetuning baseline. We have demonstrated that our best 33B model trained on the Open Assistant dataset can rival ChatGPT on the Vicuna benchmark. Since instruction finetuning is an essential tool to transform raw pretrained LLMs into ChatGPT-like chatbots, we believe that our method will make finetuning widespread and common in particular for the researchers that have the least resources, a big win for the accessibility of state of the art NLP technology. QLoRA can be seen as an equalizing factor that helps to close the resource gap between large corporations and small teams with consumer GPUs.

Another potential source of impact is deployment to mobile phones. We believe our QLoRA method might enable the critical milestone of enabling the finetuning of LLMs on phones and other low resource settings. While 7B models were shown to be able to be run on phones before, QLoRA is the first method that would enable the finetuning of such models. We estimate that with an iPhone 12 Plus, QLoRA can finetune 3 million tokens per night while the phone is charging. While finetuned 7B models do not reach the quality of ChatGPT, we believe that the quality is good enough to enable novel applications that have not been possible before due to privacy or LLM quality issues. QLoRA can help enable privacy-preserving usage of LLMs, where users can own and manage their own data and models, while simultaneously making LLMs easier to deploy.

However, finetuning is a dual-use technology that can be abused to cause harm. Widespread use of LLMs has known dangers [Bommasani et al., 2021; Bender et al., 2021], but we believe that equalizing access to a technology that is quickly becoming ubiquitous will allow for better more independent analysis than keeping the power of LLMs in the hands of large corporations that do not release models or source code for auditing.

All in all, we believe that QLoRA will have a broadly positive impact making the finetuning of high quality LLMs much more widely and easily accessible.

4.10 Conclusion

In this chapter, we show how NF4 4-bit quantization combined with parameter efficient finetuning can dramatically reduce the memory footprint of supervised finetuning for post-training without performance degradation over standard full-precision finetuning. On the inference side, quantization can introduces approximations that reduce performance compared to the full precision model. The low rank adapters in

QLoRA help correct the quantization error with lightweight finetuning on relevant instruction tuning data minimizing quantization errors while improving the inference memory requirements. In summary, this chapter demonstrates how quantization and parameter efficient finetuning can help scale model parameters while containing memory requirements with minimal performance tradeoffs.

Chapter 5

Conclusion

5.1 Conclusion

This dissertation has demonstrated that sustainable scaling of large language models requires fundamentally rethinking efficiency at each stage of the development pipeline—from architecture and pretraining through mid-training to post-training and inference. Through three complementary innovations, we have shown that it is possible to extract significantly more capability per unit of computational resource without sacrificing performance.

The Byte Latent Transformer (BLT) challenges the assumption that tokenization is necessary for efficient language modeling. By introducing dynamic, entropy-based patching that allocates compute based on prediction complexity, BLT achieves up to 50% reduction in inference FLOPs while matching the performance of tokenization-based models. More importantly, BLT's architecture unlocks a new scaling dimension where both model size and patch size can be increased simultaneously within a fixed inference budget, demonstrating that architectural innovations can fundamentally improve scaling trends.

Socratic Pretraining addresses the critical challenge of data efficiency in the mid-training phase. By transforming unlabeled web documents into rich question-answer supervision, this approach demonstrates that careful synthetic augmentation and improved training objectives can halve the requirement for expensive task-specific labeled data. The success of Socratic Pretraining on controllable summarization tasks underscores a broader principle: the quality of training signal matters more than quantity, and synthetic aug-

mentation guided by task structure can unlock capabilities that would otherwise require slow and expensive human annotation.

QLoRA proves that the performance-precision trade-off in neural networks is not as rigid as previously believed. By combining 4-bit quantization with low-rank adaptation, we reduce the memory footprint of finetuning by 15× without performance degradation, democratizing access to state-of-the-art language models. The extensive instruction tuning studies enabled by QLoRA's efficiency reveal that data quality dominates dataset size for finetuning downstream performance—a finding that further reinforces the importance of thoughtful data curation over simple scaling.

Together, these contributions paint a picture of a more sustainable future for language model development. Rather than accepting exponential growth in compute, data, and energy as inevitable, we have shown that intelligent architectural, data centric, and algorithmic choices can improve the scaling trends themselves. The practical impact is already evident: BLT enables more efficient pretraining and deployment, Socratic Pretraining exemplifies synthetic data augmentation reducing reliance on scarce human annotations, and QLoRA has made it possible to finetune 65B parameter models on consumer hardware, spawning thousands of community-driven model variants.

Perhaps most importantly, this work demonstrates that efficiency and capability are not opposing forces for LLMs. Each method in this dissertation achieves its efficiency gains not through crude approximation but through more intelligent use of available resources—whether by allocating compute based on entropy, extracting richer supervision from existing data, or correcting quantization errors through targeted adaptation. This suggests a broader principle for future work: the path to more capable AI systems may lie not in consuming ever more resources, but in more carefully allocating the resources we have. As language models continue to reshape how humanity interacts with information, advancing their efficiency becomes our imperative to unlock unprecedented capabilities and democratize access to transformative AI.

5.2 Future Work

Efficiency scaling techniques developed in this dissertation open several promising avenues for future research.

Towards Efficient Omni Model Architectures The dynamic patching approach of BLT, combined with its byte-level modeling capabilities, suggests natural extensions to domains beyond text where encoding challenges, sequence length, and computational efficiency pose critical bottlenecks. One particularly compelling direction is the application of entropy-based patching to speech processing. Current state-of-the-art speech tokenizers compress audio to approximately 20 tokens per second, creating a computational barrier for deploying large language models in real-time speech applications. By adapting BLT's patching strategy to compress speech signals down to 1-2 patches per second, we could potentially enable real-time deployment of much larger models for speech recognition, translation, and synthesis tasks. The key challenge lies in determining optimal patching strategies.

Similar opportunities exist in protein modeling, robotics, and video understanding, where long sequences currently limit the application of large transformer models. For proteins, patching could group amino acids based on structural or functional similarity rather than sequential proximity. In robotics, sensor data streams could be compressed into meaningful action primitives. Each domain presents unique challenges in defining what constitutes an appropriate "patch" and how to maintain critical information during compression.

The natural evolution of these ideas points toward end-to-end learned patching, where the segmentation strategy itself is optimized during training rather than predetermined. This could involve differentiable patching mechanisms that learn to segment sequences optimally for the downstream task, potentially discovering structure that fixed heuristics miss. Such learned patching could adapt dynamically not just to the complexity of different parts of the input, but also to the specific requirements of the task at hand.

Efficient Scaling of Data Quality and Quantity The synergy between pretraining, post-training, and inference stages presents rich opportunities for improving data efficiency. A particularly promising direction, based on the work in this thesis, is data rewriting, where existing datasets are systematically transformed to enhance their training signal. Rather than just collecting more data, we can leverage language models to rewrite existing corpora with improved clarity, diversity, or task-specific structure. This approach could increase the effective size of our training data without requiring new human-authored content.

As we approach the limits of high-quality human-generated text, synthetic data generation and curation becomes critical for continued scaling. However, this must be approached systematically through a staged

process that creates a virtuous cycle between model capabilities and data quality. Models enhanced through RL post-training develop new capabilities that can be leveraged to generate higher-quality synthetic data. This improved data can then feed back into pretraining future model generations, making them more likely to acquire even more advanced capabilities. This iterative refinement process suggests that the traditional separation between pretraining, post-training, and inference may need to give way to a more integrated development cycle where each stage informs the others.

The key challenge lies in ensuring that synthetic data maintains diversity and avoids mode collapse while systematically improving upon human-generated text in targeted dimensions such as reasoning clarity, contextual factual accuracy, or pedagogical value. Future work should develop principled methods for this data-model co-evolution, potentially drawing on ideas from curriculum learning and active learning to prioritize the generation of data that maximally improves model capabilities.

Scaling Post-Training and Inference Sparsity emerges as a crucial technique that complements the quantization approaches demonstrated in QLoRA. By learning to activate only relevant model components for each input, we can achieve substantial computational savings during both training and inference. This selective activation pattern could be learned during post-training and refined during inference, creating models that dynamically allocate their capacity based on task complexity.

The success of quantization techniques like QLoRA in post-training is now extending into pretraining itself. Recent work [Liu et al., 2024] suggests that training directly with reduced precision from the start, rather than quantizing after training, may yield to more efficient training without significant performance tradeoffs. This could eliminate the performance gap between full and reduced precision models while helping alleviate memory and computational bottlenecks during pretraining stages.

The emerging paradigm of test-time compute scaling, where models perform extended reasoning before generating responses, further emphasizes the importance of efficient architectures. As models increasingly "think" before they respond, the ability to perform this reasoning efficiently becomes paramount. Working in the latent space, as BLT does through its patch representations, could offer significant advantages. As models increasingly employ extended reasoning before generating answers, the ability to manipulate compact, high-level patch representations becomes crucial. These patches could serve as the "concepts" that models reason over during test-time computation, enabling more efficient search and planning algorithms that explore a

broader solution space without being constrained by token-level granularity.

More broadly, increasingly diverse usecases might benefit from dynamic compute allocation that adapts to the complexity of each prediction. Just as BLT demonstrated with entropy-based patching at the sequence level, we need mechanisms that can vary compute on multiple axes like depth, width, and token sequence timescales. During inference, this could manifest as adaptive depth networks that use fewer layers for simple predictions, or mixture-of-experts architectures that route different inputs to specialized subnetworks. In post-training, this could mean learning to identify which model components are most important for specific tasks and selectively updating only those parameters.

Summary Finally, the principles demonstrated in this dissertation—that thoughtful algorithmic design can overcome hardware limitations, that quality trumps quantity in data, and that efficiency enables rather than constrains capability—suggest a broader research agenda. As we approach physical limits in semiconductor scaling and data availability, the techniques for extracting more intelligence per unit resource will become increasingly critical. The next breakthroughs in AI may come not from building larger models or datasets, but from discovering fundamentally more efficient ways to represent, process, and learn from information.

Bibliography

Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. 2023. Gpt-4 technical report. arXiv preprint arXiv:2303.08774.

Ester Aflalo. 2021. Students generating questions as a way of learning. Active Learning in Higher Education, 22(1):63–75.

Divyansh Agarwal, Alexander R. Fabbri, Simeng Han, Wojciech Kryscinski, Faisal Ladhak, Bryan Li, Kathleen McKeown, Dragomir Radev, Tianyi Zhang, and Sam Wiseman. 2022. CREATIVESUMM: Shared task on automatic summarization for creative writing. In Proceedings of the Workshop on Automatic Summarization for Creative Writing, pages 67–73, Gyeongju, Republic of Korea. Association for Computational Linguistics.

Armen Aghajanyan, Anchit Gupta, Akshat Shrivastava, Xilun Chen, Luke Zettlemoyer, and Sonal Gupta. 2021. Muppet: Massive multi-task representations with pre-finetuning. In Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing, pages 5799–5811, Online and Punta Cana, Dominican Republic. Association for Computational Linguistics.

Rami Al-Rfou, Dokook Choe, Noah Constant, Mandy Guo, and Llion Jones. 2019. Character-level language modeling with deeper self-attention. In Proceedings of the Thirty-Third AAAI Conference on Artificial Intelligence and Thirty-First Innovative Applications of Artificial Intelligence Conference and Ninth AAAI Symposium on Educational Advances in Artificial Intelligence, AAAI'19/IAAI'19/EAAI'19. AAAI Press.

Chris Alberti, Daniel Andor, Emily Pitler, Jacob Devlin, and Michael Collins. 2019. Synthetic QA corpora generation with roundtrip consistency. In Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics, pages 6168–6173, Florence, Italy. Association for Computational Linguistics.

Shengnan An, Yifei Li, Zeqi Lin, Qian Liu, Bei Chen, Qiang Fu, Weizhu Chen, Nanning Zheng, and Jian-Guang Lou. 2022. Input-tuning: Adapting unfamiliar inputs to frozen pretrained models. arXiv preprint arXiv:2203.03131.

Amanda Askell, Yuntao Bai, Anna Chen, Dawn Drain, Deep Ganguli, Tom Henighan, Andy Jones, Nicholas Joseph, Ben Mann, Nova DasSarma, et al. 2021. A general language assistant as a laboratory for alignment. arXiv preprint arXiv:2112.00861.

Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, et al. 2021. Program synthesis with large language models. arXiv preprint arXiv:2108.07732.

Stephen H Bach, Victor Sanh, Zheng-Xin Yong, Albert Webson, Colin Raffel, Nihal V Nayak, Abheesht Sharma, Taewoon Kim, M Saiful Bari, Thibault Fevry, et al. 2022. Promptsource: An integrated development environment and repository for natural language prompts. arXiv preprint arXiv:2202.01279.

Bing Bai, Jason Weston, David Grangier, Ronan Collobert, Kunihiko Sadamasa, Yanjun Qi, Olivier Chapelle, and Kilian Weinberger. 2010. Learning to rank with (a lot of) word features. Information retrieval, 13:291–314.

Yuntao Bai, Andy Jones, Kamal Ndousse, Amanda Askell, Anna Chen, Nova DasSarma, Dawn Drain, Stanislav Fort, Deep Ganguli, Tom Henighan, et al. 2022a. Training a helpful and harmless assistant with reinforcement learning from human feedback. arXiv preprint arXiv:2204.05862.

Yuntao Bai, Saurav Kadavath, Sandipan Kundu, Amanda Askell, Jackson Kernion, Andy Jones, Anna Chen, Anna Goldie, Azalia Mirhoseini, Cameron McKinnon, et al. 2022b. Constitutional ai: Harmlessness from ai feedback. arXiv preprint arXiv:2212.08073.

Emily M Bender, Timnit Gebru, Angelina McMillan-Major, and Shmargaret Shmitchell. 2021. On the

dangers of stochastic parrots: Can language models be too big? In Proceedings of the 2021 ACM conference on fairness, accountability, and transparency, pages 610–623.

Stella Biderman, Hailey Schoelkopf, Quentin Anthony, Herbie Bradley, Kyle O'Brien, Eric Hallahan, Mohammad Aflah Khan, Shivanshu Purohit, USVSN Sai Prashanth, Edward Raff, et al. 2023. Pythia: A suite for analyzing large language models across training and scaling. arXiv preprint arXiv:2304.01373.

Yonatan Bisk, Rowan Zellers, Jianfeng Gao, Yejin Choi, et al. 2020. Piqa: Reasoning about physical commonsense in natural language. In Proceedings of the AAAI conference on artificial intelligence, volume 34, pages 7432–7439.

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

Jeremy J Bornstein, Douglass R Cutting, John D Hatton, and Daniel E Rose. 1999. Interactive document summarization. US Patent 5,867,164.

Adam Casson. 2023. Transformer flops. Adam Casson Blog.

Tuhin Chakrabarty, Justin Lewis, and Smaranda Muresan. 2022. Consistent: Open-ended question generation from news articles. arXiv preprint arXiv:2210.11536.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde De Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. 2021. Evaluating large language models trained on code. arXiv preprint arXiv:2107.03374.

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

Wei-Lin Chiang, Zhuohan Li, Zi Lin, Ying Sheng, Zhanghao Wu, Hao Zhang, Lianmin Zheng, Siyuan Zhuang, Yonghao Zhuang, Joseph E. Gonzalez, Ion Stoica, and Eric P. Xing. 2023. Vicuna: An open-source chatbot impressing gpt-4 with 90%* chatgpt quality.

Dokook Choe, Rami Al-Rfou, Mandy Guo, Heeyoung Lee, and Noah Constant. 2019. Bridging the gap for tokenizer-free language models. arXiv preprint arXiv:1908.10322.

Paul F Christiano, Jan Leike, Tom Brown, Miljan Martic, Shane Legg, and Dario Amodei. 2017. Deep reinforcement learning from human preferences. Advances in neural information processing systems, 30.

Hyung Won Chung, Le Hou, Shayne Longpre, Barret Zoph, Yi Tay, William Fedus, Eric Li, Xuezhi Wang, Mostafa Dehghani, Siddhartha Brahma, et al. 2022. Scaling instruction-finetuned language models. arXiv preprint arXiv:2210.11416.

Junyoung Chung, Sungjin Ahn, and Yoshua Bengio. 2017. Hierarchical multiscale recurrent neural networks. In 5th International Conference on Learning Representations.

Jonathan H Clark, Dan Garrette, Iulia Turc, and John Wieting. 2022. Canine: Pre-training an efficient tokenization-free encoder for language representation. Transactions of the Association for Computational Linguistics, 10:73–91.

Peter Clark, Isaac Cowhey, Oren Etzioni, Tushar Khot, Ashish Sabharwal, Carissa Schoenick, and Oyvind Tafjord. 2018. Think you have solved question answering? try arc, the ai2 reasoning challenge. arXiv preprint arXiv:1803.05457.

Richard Csaky and Gábor Recski. 2021. The Gutenberg dialogue dataset. In Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: Main Volume, pages 138–159, Online. Association for Computational Linguistics.

Gautier Dagan, Gabriel Synnaeve, and Baptiste Roziere. 2024. Getting the most out of your tokenizer for pre-training and domain adaptation. In Forty-first International Conference on Machine Learning.

Zhuyun Dai, Arun Tejasvi Chaganty, Vincent Y Zhao, Aida Amini, Qazi Mamunur Rashid, Mike Green, and Kelvin Guu. 2022. Dialog inpainting: Turning documents into dialogs. In International Conference on Machine Learning, pages 4558–4586. PMLR.

Tri Dao, Dan Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. 2022. Flashattention: Fast and

memory-efficient exact attention with io-awareness. Advances in neural information processing systems, 35:16344–16359.

Tim Dettmers, Mike Lewis, Younes Belkada, and Luke Zettlemoyer. 2022a. Gpt3.int8(): 8-bit matrix multiplication for transformers at scale. In Advances in Neural Information Processing Systems, volume 35, pages 30318–30332. Curran Associates, Inc.

Tim Dettmers, Mike Lewis, Sam Shleifer, and Luke Zettlemoyer. 2022b. 8-bit optimizers via block-wise quantization. 9th International Conference on Learning Representations, ICLR.

Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. 2023. Qlora: Efficient finetuning of quantized llms. Advances in neural information processing systems, 36:10088–10115.

Tim Dettmers and Luke Zettlemoyer. 2023. The case for 4-bit precision: k-bit inference scaling laws. In Proceedings of the 40th International Conference on Machine Learning, ICML'23. JMLR.org.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. arXiv preprint arXiv:1810.04805.

Zi-Yi Dou, Pengfei Liu, Hiroaki Hayashi, Zhengbao Jiang, and Graham Neubig. 2021. GSum: A general framework for guided neural abstractive summarization. In Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, pages 4830–4842, Online. Association for Computational Linguistics.

Xinya Du, Junru Shao, and Claire Cardie. 2017. Learning to ask: Neural question generation for reading comprehension. In Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), pages 1342–1352, Vancouver, Canada. Association for Computational Linguistics.

Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, et al. 2024. The llama 3 herd of models. arXiv preprint arXiv:2407.21783.

Lukas Edman, Helmut Schmid, and Alexander Fraser. 2024. CUTE: Measuring LLMs' understanding of their tokens. In Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing, pages 3017–3026, Miami, Florida, USA. Association for Computational Linguistics.

Angela Edmunds and Anne Morris. 2000. The problem of information overload in business organisations: a review of the literature. International journal of information management, 20(1):17–28.

Hicham El Boukkouri, Olivier Ferret, Thomas Lavergne, Hiroshi Noji, Pierre Zweigenbaum, and Jun'ichi Tsujii. 2020. CharacterBERT: Reconciling ELMo and BERT for word-level open-vocabulary representations from characters. In Proceedings of the 28th International Conference on Computational Linguistics, pages 6903–6915, Barcelona, Spain (Online). International Committee on Computational Linguistics.

Arpad E Elo. 1967. The proposed uscf rating system. its development, theory, and applications. Chess Life, 22(8):242–247.

Arpad E Elo. 1978. The rating of chessplayers, past and present. Arco Pub.

Tyna Eloundou, Sam Manning, Pamela Mishkin, and Daniel Rock. 2024. Gpts are gpts: Labor market impact potential of llms. Science, 384(6702):1306–1308.

Alexander Fabbri, Simeng Han, Haoyuan Li, Haoran Li, Marjan Ghazvininejad, Shafiq Joty, Dragomir Radev, and Yashar Mehdad. 2021a. Improving zero and few-shot abstractive summarization with intermediate fine-tuning and data augmentation. In Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, pages 704–717, Online. Association for Computational Linguistics.

Alexander R. Fabbri, Wojciech Kryściński, Bryan McCann, Caiming Xiong, Richard Socher, and Dragomir Radev. 2021b. SummEval: Re-evaluating summarization evaluation. Transactions of the Association for Computational Linguistics, 9:391–409.

Angela Fan, David Grangier, and Michael Auli. 2018. Controllable abstractive summarization. In Proceedings of the 2nd Workshop on Neural Machine Translation and Generation, pages 45–54, Melbourne, Australia. Association for Computational Linguistics.

Joseph L Fleiss. 1971. Measuring nominal scale agreement among many raters. Psychological bulletin, 76(5):378.

Elias Frantar, Saleh Ashkboos, Torsten Hoefler, and Dan Alistarh. 2022. Gptq: Accurate post-training quantization for generative pre-trained transformers. arXiv preprint arXiv:2210.17323.

Jinlan Fu, See-Kiong Ng, Zhengbao Jiang, and Pengfei Liu. 2023. Gptscore: Evaluate as you desire. arXiv preprint arXiv:2302.04166.

Philip Gage. 1994. A new algorithm for data compression. The C Users Journal, 12(2):23–38.

Leo Gao, Stella Biderman, Sid Black, Laurence Golding, Travis Hoppe, Charles Foster, Jason Phang, Horace He, Anish Thite, Noa Nabeshima, et al. 2020. The pile: An 800gb dataset of diverse text for language modeling. arXiv preprint arXiv:2101.00027.

Sebastian Gehrmann, Tosin Adewumi, Karmanya Aggarwal, Pawan Sasanka Ammanamanchi, Anuoluwapo Aremu, Antoine Bosselut, Khyathi Raghavi Chandu, Miruna-Adriana Clinciu, Dipanjan Das, Kaustubh Dhole, Wanyu Du, Esin Durmus, Ondřej Dušek, Chris Chinenye Emezue, Varun Gangal, Cristina Garbacea, Tatsunori Hashimoto, Yufang Hou, Yacine Jernite, Harsh Jhamtani, Yangfeng Ji, Shailza Jolly, Mihir Kale, Dhruv Kumar, Faisal Ladhak, Aman Madaan, Mounica Maddela, Khyati Mahajan, Saad Mahamood, Bodhisattwa Prasad Majumder, Pedro Henrique Martins, Angelina McMillan-Major, Simon Mille, Emiel van Miltenburg, Moin Nadeem, Shashi Narayan, Vitaly Nikolaev, Andre Niyongabo Rubungo, Salomey Osei, Ankur Parikh, Laura Perez-Beltrachini, Niranjan Ramesh Rao, Vikas Raunak, Juan Diego Rodriguez, Sashank Santhanam, João Sedoc, Thibault Sellam, Samira Shaikh, Anastasia Shimorina, Marco Antonio Sobrevilla Cabezudo, Hendrik Strobelt, Nishant Subramani, Wei Xu, Diyi Yang, Akhila Yerukola, and Jiawei Zhou. 2021. The GEM benchmark: Natural language generation, its evaluation and metrics. In Proceedings of the 1st Workshop on Natural Language Generation, Evaluation, and Metrics (GEM 2021), pages 96–120, Online. Association for Computational Linguistics.

Xinyang Geng, Arnav Gudibande, Hao Liu, Eric Wallace, Pieter Abbeel, Sergey Levine, and Dawn Song. 2023. Koala: A dialogue model for academic research. Blog post.

Amelia Glaese, Nat McAleese, Maja Trębacz, John Aslanides, Vlad Firoiu, Timo Ewalds, Maribeth Rauh, Laura Weidinger, Martin Chadwick, Phoebe Thacker, et al. 2022. Improving alignment of dialogue agents via targeted human judgements. arXiv preprint arXiv:2209.14375.

Naman Goyal, Cynthia Gao, Vishrav Chaudhary, Peng-Jen Chen, Guillaume Wenzek, Da Ju, Sanjana Krishnan, Marc'Aurelio Ranzato, Francisco Guzmán, and Angela Fan. 2022a. The Flores-101 evaluation benchmark for low-resource and multilingual machine translation. Transactions of the Association for Computational Linguistics, 10:522–538.

Tanya Goyal, Junyi Jessy Li, and Greg Durrett. 2022b. News summarization and evaluation in the era of gpt-3. arXiv preprint arXiv:2209.12356.

Alex Graves. 2013. Generating sequences with recurrent neural networks. arXiv preprint arXiv:1308.0850.

Albert Gu and Tri Dao. 2024. Mamba: Linear-time sequence modeling with selective state spaces. In First Conference on Language Modeling.

Mandy Guo, Joshua Ainslie, David Uthus, Santiago Ontanon, Jianmo Ni, Yun-Hsuan Sung, and Yinfei Yang. 2022. LongT5: Efficient text-to-text transformer for long sequences. In Findings of the Association for Computational Linguistics: NAACL 2022, pages 724–736, Seattle, United States. Association for Computational Linguistics.

Suchin Gururangan, Ana Marasović, Swabha Swayamdipta, Kyle Lo, Iz Beltagy, Doug Downey, and Noah A. Smith. 2020. Don't stop pretraining: Adapt language models to domains and tasks. In Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, pages 8342–8360, Online. Association for Computational Linguistics.

Suchin Gururangan, Swabha Swayamdipta, Omer Levy, Roy Schwartz, Samuel R Bowman, and Noah A Smith. 2018. Annotation artifacts in natural language inference data. arXiv preprint arXiv:1803.02324.

Junxian He, Wojciech Kryscinski, Bryan McCann, Nazneen Rajani, and Caiming Xiong. 2022a. CTRLsum: Towards generic controllable text summarization. In Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing, pages 5879–5915, Abu Dhabi, United Arab Emirates. Association for Computational Linguistics.

Wanwei He, Yinpei Dai, Yinhe Zheng, Yuchuan Wu, Zheng Cao, Dermot Liu, Peng Jiang, Min Yang, Fei Huang, Luo Si, et al. 2022b. Galaxy: A generative pre-trained model for task-oriented dialog with semi-supervised learning and explicit policy injection. Proceedings of the AAAI Conference on Artificial Intelligence.

James Henderson, Sebastian Ruder, et al. 2021. Compacter: Efficient low-rank hypercomplex adapter layers. In Advances in Neural Information Processing Systems.

Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. 2020. Measuring massive multitask language understanding. In International Conference on Learning Representations.

Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. 2021. Measuring massive multitask language understanding. In International Conference on Learning Representations.

Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. 2015. Distilling the knowledge in a neural network. arXiv preprint arXiv:1503.02531.

Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, Tom Hennigan, Eric Noland, Katherine Millican, George van den Driessche, Bogdan Damoc, Aurelia Guy, Simon Osindero, Karen Simonyan, Erich Elsen, Oriol Vinyals, Jack William Rae, and Laurent Sifre. 2022. An empirical analysis of compute-optimal large language model training. In Advances in Neural Information Processing Systems.

Ari Holtzman, Jan Buys, Li Du, Maxwell Forbes, and Yejin Choi. 2020. The curious case of neural text degeneration. In International Conference on Learning Representations.

Or Honovich, Thomas Scialom, Omer Levy, and Timo Schick. 2022. Unnatural instructions: Tuning language models with (almost) no human labor. arXiv preprint arXiv:2212.09689.

Neil Houlsby, Andrei Giurgiu, Stanislaw Jastrzebski, Bruna Morrone, Quentin De Laroussilhe, Andrea Gesmundo, Mona Attariyan, and Sylvain Gelly. 2019. Parameter-efficient transfer learning for nlp. In International Conference on Machine Learning, pages 2790–2799. PMLR.

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

Zhiting Hu, Zichao Yang, Xiaodan Liang, Ruslan Salakhutdinov, and Eric P. Xing. 2017. Toward controlled generation of text. In Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017, volume 70 of Proceedings of Machine Learning Research, pages 1587–1596. PMLR.

Srinivasan Iyer, Xi Victoria Lin, Ramakanth Pasunuru, Todor Mihaylov, Dániel Simig, Ping Yu, Kurt Shuster, Tianlu Wang, Qing Liu, Punit Singh Koura, et al. 2022. Opt-iml: Scaling language model instruction meta learning through the lens of generalization. arXiv preprint arXiv:2212.12017.

Andrew Jaegle, Felix Gimeno, Andy Brock, Oriol Vinyals, Andrew Zisserman, and Joao Carreira. 2021. Perceiver: General perception with iterative attention. PMLR.

Robin Jia, Mike Lewis, and Luke Zettlemoyer. 2022. Question answering infused pre-training of general-purpose contextualized representations. In Findings of the Association for Computational Linguistics: ACL 2022, pages 711–728, Dublin, Ireland. Association for Computational Linguistics.

Nal Kalchbrenner, Lasse Espeholt, Karen Simonyan, Aaron van den Oord, Alex Graves, and Koray Kavukcuoglu. 2016. Neural machine translation in linear time. arXiv preprint arXiv:1610.10099.

Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. 2020. Scaling laws for neural language models. arXiv preprint arXiv:2001.08361.

Tom Kenter, Llion Jones, and Daniel Hewlett. 2018. Byte-level machine reading across morphologically varied languages. In Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence and Thirtieth Innovative Applications of Artificial Intelligence Conference and Eighth AAAI Symposium on Educational Advances in Artificial Intelligence, AAAI'18/IAAI'18/EAAI'18. AAAI Press.

Yoon Kim, Yacine Jernite, David Sontag, and Alexander M. Rush. 2016. Character-aware neural language

models. In Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence, AAAI'16, page 2741–2749. AAAI Press.

Wei-Jen Ko, Cutter Dalton, Mark Simmons, Eliza Fisher, Greg Durrett, and Junyi Jessy Li. 2021. Discourse comprehension: A question answering framework to represent sentence connections. arXiv preprint arXiv:2111.00701.

Abdullatif Köksal, Timo Schick, Anna Korhonen, and Hinrich Schütze. 2023. Longform: Optimizing instruction tuning for long text generation with corpus extraction. arXiv preprint arXiv:2304.08460.

Andreas Köpf, Yannic Kilcher, Dimitri von Rütte, Sotiris Anagnostidis, Zhi-Rui Tam, Keith Stevens, Abdullah Barhoum, Nguyen Minh Duc, Oliver Stanley, Richárd Nagyfi, et al. 2023. Openassistant conversations–democratizing large language model alignment. arXiv preprint arXiv:2304.07327.

Sayali Kulkarni, Sheide Chammas, Wan Zhu, Fei Sha, and Eugene Ie. 2020. Aquamuse: Automatically generating datasets for query-based multi-document summarization. arXiv preprint arXiv:2010.12694.

Sachin Kumar, Eric Malmi, Aliaksei Severyn, and Yulia Tsvetkov. 2021. Controlled text generation as continuous optimization with multiple constraints. Advances in Neural Information Processing Systems, 34:14542–14554.

LAION. 2023. Open-instruction-generalist dataset. `https://github.com/LAION-AI/Open-Instruction-Generalist`.

Brian Lester, Rami Al-Rfou, and Noah Constant. 2021. The power of scale for parameter-efficient prompt tuning. arXiv preprint arXiv:2104.08691.

Brian Lester, Jaehoon Lee, Alex Alemi, Jeffrey Pennington, Adam Roberts, Jascha Sohl-Dickstein, and Noah Constant. 2024. Training llms over neurally compressed text. arXiv preprint arXiv:2404.03626.

Anton Leuski, Chin-Yew Lin, and Eduard Hovy. 2003. iNeATS: Interactive multi-document summarization. In The Companion Volume to the Proceedings of 41st Annual Meeting of the Association for Computational Linguistics, pages 125–128, Sapporo, Japan. Association for Computational Linguistics.

Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Veselin Stoyanov, and Luke Zettlemoyer. 2020. BART: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. In Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, pages 7871–7880, Online. Association for Computational Linguistics.

Jeffrey Li, Alex Fang, Georgios Smyrnis, Maor Ivgi, Matt Jordan, Samir Yitzhak Gadre, Hritik Bansal, Etash Kumar Guha, Sedrick Keh, Kushal Arora, Saurabh Garg, Rui Xin, Niklas Muennighoff, Reinhard Heckel, Jean Mercat, Mayee F Chen, Suchin Gururangan, Mitchell Wortsman, Alon Albalak, Yonatan Bitton, Marianna Nezhurina, Amro Kamal Mohamed Abbas, Cheng-Yu Hsieh, Dhruba Ghosh, Joshua P Gardner, Maciej Kilian, Hanlin Zhang, Rulin Shao, Sarah M Pratt, Sunny Sanyal, Gabriel Ilharco, Giannis Daras, Kalyani Marathe, Aaron Gokaslan, Jieyu Zhang, Khyathi Chandu, Thao Nguyen, Igor Vasiljevic, Sham M. Kakade, Shuran Song, Sujay Sanghavi, Fartash Faghri, Sewoong Oh, Luke Zettlemoyer, Kyle Lo, Alaaeldin El-Nouby, Hadi Pouransari, Alexander T Toshev, Stephanie Wang, Dirk Groeneveld, Luca Soldaini, Pang Wei Koh, Jenia Jitsev, Thomas Kollar, Alex Dimakis, Yair Carmon, Achal Dave, Ludwig Schmidt, and Vaishaal Shankar. 2024. Datacomp-LM: In search of the next generation of training sets for language models. In The Thirty-eight Conference on Neural Information Processing Systems Datasets and Benchmarks Track.

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

Davis Liang, Hila Gonen, Yuning Mao, Rui Hou, Naman Goyal, Marjan Ghazvininejad, Luke Zettlemoyer, and Madian Khabsa. 2023. XLM-V: Overcoming the vocabulary bottleneck in multilingual masked language models. In Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing, pages 13142–13152, Singapore. Association for Computational Linguistics.

Percy Liang, Rishi Bommasani, Tony Lee, Dimitris Tsipras, Dilara Soylu, Michihiro Yasunaga, Yian Zhang, Deepak Narayanan, Yuhuai Wu, Ananya Kumar, et al. 2022. Holistic evaluation of language models. arXiv preprint arXiv:2211.09110.

Thomas Liao, Rohan Taori, Inioluwa Deborah Raji, and Ludwig Schmidt. 2021. Are we learning yet? a meta

review of evaluation failures across machine learning. In Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 2).

Tomasz Limisiewicz, Terra Blevins, Hila Gonen, Orevaoghene Ahia, and Luke Zettlemoyer. 2024. MYTE: Morphology-driven byte encoding for better and fairer multilingual language modeling. In Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), pages 15059–15076, Bangkok, Thailand. Association for Computational Linguistics.

Chin-Yew Lin. 2004. ROUGE: A package for automatic evaluation of summaries. In Text Summarization Branches Out, pages 74–81, Barcelona, Spain. Association for Computational Linguistics.

Aixin Liu, Bei Feng, Bing Xue, Bingxuan Wang, Bochao Wu, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, et al. 2024. Deepseek-v3 technical report. arXiv preprint arXiv:2412.19437.

Haokun Liu, Derek Tam, Mohammed Muqeeth, Jay Mohta, Tenghao Huang, Mohit Bansal, and Colin A Raffel. 2022a. Few-shot parameter-efficient fine-tuning is better and cheaper than in-context learning. Advances in Neural Information Processing Systems, 35:1950–1965.

Peter J. Liu, Mohammad Saleh, Etienne Pot, Ben Goodrich, Ryan Sepassi, Lukasz Kaiser, and Noam Shazeer. 2018. Generating wikipedia by summarizing long sequences. In 6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings. OpenReview.net.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. Roberta: A robustly optimized bert pretraining approach. arXiv preprint arXiv:1907.11692.

Yixin Liu, Alexander R Fabbri, Pengfei Liu, Yilun Zhao, Linyong Nan, Ruilin Han, Simeng Han, Shafiq Joty, Chien-Sheng Wu, Caiming Xiong, et al. 2022b. Revisiting the gold standard: Grounding summarization evaluation with robust human evaluation. arXiv preprint arXiv:2212.07981.

Shayne Longpre, Le Hou, Tu Vu, Albert Webson, Hyung Won Chung, Yi Tay, Denny Zhou, Quoc V Le,

Barret Zoph, Jason Wei, et al. 2023. The flan collection: Designing data and methods for effective instruction tuning. arXiv preprint arXiv:2301.13688.

Ilya Loshchilov and Frank Hutter. 2019. Decoupled weight decay regularization. In International Conference on Learning Representations.

Tomáš Mikolov, Ilya Sutskever, Anoop Deoras, Hai-Son Le, Stefan Kombrink, and Jan Cernocky. 2012. Subword language modeling with neural networks. preprint (http://www. fit. vutbr. cz/imikolov/rnnlm/char. pdf), 8(67).

Sewon Min, Mike Lewis, Luke Zettlemoyer, and Hannaneh Hajishirzi. 2021. Metaicl: Learning to learn in context. arXiv preprint arXiv:2110.15943.

Lidiya Murakhovs'ka, Chien-Sheng Wu, Philippe Laban, Tong Niu, Wenhao Liu, and Caiming Xiong. 2022. MixQG: Neural question generation with mixed answer types. In Findings of the Association for Computational Linguistics: NAACL 2022, pages 1486–1497, Seattle, United States. Association for Computational Linguistics.

Shashi Narayan, Joshua Maynez, Reinald Kim Amplayo, Kuzman Ganchev, Annie Louis, Fantine Huot, Dipanjan Das, and Mirella Lapata. 2022. Conditional generation with a question-answering blueprint. ArXiv preprint, abs/2207.00397.

Shashi Narayan, Gonçalo Simoes, Ji Ma, Hannah Craighead, and Ryan Mcdonald. 2020. Qurious: Question generation pretraining for text generation. arXiv preprint arXiv:2004.11026.

Shashi Narayan, Yao Zhao, Joshua Maynez, Gonçalo Simões, Vitaly Nikolaev, and Ryan McDonald. 2021. Planning with learned entity prompts for abstractive summarization. Transactions of the Association for Computational Linguistics, 9:1475–1492.

Piotr Nawrot, Jan Chorowski, Adrian Lancucki, and Edoardo Maria Ponti. 2023. Efficient transformers with dynamic token pooling. In Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), pages 6403–6417, Toronto, Canada. Association for Computational Linguistics.

Piotr Nawrot, Szymon Tworkowski, Michał Tyrolski, Lukasz Kaiser, Yuhuai Wu, Christian Szegedy, and Henryk Michalewski. 2022. Hierarchical transformers are more efficient language models. In Findings of the Association for Computational Linguistics: NAACL 2022, pages 1559–1571, Seattle, United States. Association for Computational Linguistics.

Aida Nematzadeh, Kaylee Burns, Erin Grant, Alison Gopnik, and Tom Griffiths. 2018. Evaluating theory of mind in question answering. In Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing, pages 2392–2400.

Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. 2022. Training language models to follow instructions with human feedback. Advances in Neural Information Processing Systems, 35:27730–27744.

Artidoro Pagnoni, Vidhisha Balachandran, and Yulia Tsvetkov. 2021. Understanding factuality in abstractive summarization with FRANK: A benchmark for factuality metrics. In Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, pages 4812–4829, Online. Association for Computational Linguistics.

Artidoro Pagnoni, Alex Fabbri, Wojciech Kryscinski, and Chien-Sheng Wu. 2023. Socratic pretraining: Question-driven pretraining for controllable summarization. In Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), pages 12737–12755, Toronto, Canada. Association for Computational Linguistics.

Artidoro Pagnoni, Ram Pasunuru, Pedro Rodriguez, John Nguyen, Benjamin Muller, Margaret Li, Chunting Zhou, Lili Yu, Jason Weston, Luke Zettlemoyer, et al. 2024. Byte latent transformer: Patches scale better than tokens. arXiv preprint arXiv:2412.09871.

Gunho Park, Baeseong Park, Se Jung Kwon, Byeongwook Kim, Youngjoo Lee, and Dongsoo Lee. 2022. nuqmm: Quantized matmul for efficient inference of large-scale generative language models. arXiv preprint arXiv:2206.09557.

Baolin Peng, Chunyuan Li, Pengcheng He, Michel Galley, and Jianfeng Gao. 2023. Instruction tuning with gpt-4. arXiv preprint arXiv:2304.03277.

Aleksandar Petrov, Emanuele La Malfa, Philip H.S. Torr, and Adel Bibi. 2023. Language model tokenizers introduce unfairness between languages. In Proceedings of the 37th International Conference on Neural Information Processing Systems, NIPS '23, Red Hook, NY, USA. Curran Associates Inc.

Konstantin F Pilz, James Sanders, Robi Rahman, and Lennart Heim. 2025. Trends in ai supercomputers. arXiv preprint arXiv:2504.16026.

Adam Poliak, Jason Naradowsky, Aparajita Haldar, Rachel Rudinger, and Benjamin Van Durme. 2018. Hypothesis only baselines in natural language inference. In Proceedings of the Seventh Joint Conference on Lexical and Computational Semantics, pages 180–191.

Reiner Pope, Sholto Douglas, Aakanksha Chowdhery, Jacob Devlin, James Bradbury, Anselm Levskaya, Jonathan Heek, Kefan Xiao, Shivani Agrawal, and Jeff Dean. 2022. Efficiently scaling transformer inference. arXiv preprint arXiv:2211.05102.

Guanghui Qin and Jason Eisner. 2021. Learning how to ask: Querying lms with mixtures of soft prompts. arXiv preprint arXiv:2104.06599.

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

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

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. 2020b. Exploring the limits of transfer learning with a unified text-to-text transformer. J. Mach. Learn. Res., 21(1).

Barak Rosenshine, Carla Meister, and Saul Chapman. 1996. Teaching students to generate questions: A review of the intervention studies. Review of Educational Research, 66(2):181–221.

Victor Sanh, Albert Webson, Colin Raffel, Stephen H Bach, Lintang Sutawika, Zaid Alyafeai, Antoine

Chaffin, Arnaud Stiegler, Teven Le Scao, Arun Raja, et al. 2021. Multitask prompted training enables zero-shot task generalization. arXiv preprint arXiv:2110.08207.

Maarten Sap, Ronan LeBras, Daniel Fried, and Yejin Choi. 2022. Neural theory-of-mind? on the limits of social intelligence in large lms. arXiv preprint arXiv:2210.13312.

Teven Le Scao, Angela Fan, Christopher Akiki, Ellie Pavlick, Suzana Ilić, Daniel Hesslow, Roman Castagné, Alexandra Sasha Luccioni, François Yvon, Matthias Gallé, et al. 2022. Bloom: A 176b-parameter open-access multilingual language model. arXiv preprint arXiv:2211.05100.

Rico Sennrich, Barry Haddow, and Alexandra Birch. 2016. Neural machine translation of rare words with subword units. In Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), pages 1715–1725, Berlin, Germany. Association for Computational Linguistics.

S Shaphiro and MBJB Wilk. 1965. An analysis of variance test for normality. Biometrika, 52(3):591–611.

Noam Shazeer. 2020. Glu variants improve transformer. arXiv preprint arXiv:2002.05202.

Kevin Slagle. 2024. Spacebyte: Towards deleting tokenization from large language modeling. In The Thirty-eighth Annual Conference on Neural Information Processing Systems.

Jianlin Su, Murtadha Ahmed, Yu Lu, Shengfeng Pan, Wen Bo, and Yunfeng Liu. 2024. Roformer: Enhanced transformer with rotary position embedding. Neurocomput., 568(C).

Yi-Lin Sung, Varun Nair, and Colin A Raffel. 2021. Training neural networks with fixed sparse masks. Advances in Neural Information Processing Systems, 34:24193–24205.

Ilya Sutskever, James Martens, and Geoffrey E Hinton. 2011. Generating text with recurrent neural networks. In Proceedings of the 28th international conference on machine learning (ICML-11), pages 1017–1024.

Rohan Taori, Ishaan Gulrajani, Tianyi Zhang, Yann Dubois, Xuechen Li, Carlos Guestrin, Percy Liang, and Tatsunori B. Hashimoto. 2023. Stanford alpaca: An instruction-following llama model. `https://github.com/tatsu-lab/stanford_alpaca`.

Yi Tay, Mostafa Dehghani, Vinh Q. Tran, Xavier Garcia, Jason Wei, Xuezhi Wang, Hyung Won Chung, Dara Bahri, Tal Schuster, Steven Zheng, Denny Zhou, Neil Houlsby, and Donald Metzler. 2023. UL2: Unifying language learning paradigms. In The Eleventh International Conference on Learning Representations.

Romal Thoppilan, Daniel De Freitas, Jamie Hall, Noam Shazeer, Apoorv Kulshreshtha, Heng-Tze Cheng, Alicia Jin, Taylor Bos, Leslie Baker, Yu Du, et al. 2022. Lamda: Language models for dialog applications. arXiv preprint arXiv:2201.08239.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. 2023. Llama: Open and efficient foundation language models. arXiv preprint arXiv:2302.13971.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. arXiv preprint arXiv:1706.03762.

Jesse Vig, Alexander Fabbri, Wojciech Kryscinski, Chien-Sheng Wu, and Wenhao Liu. 2022. Exploring neural models for query-focused summarization. In Findings of the Association for Computational Linguistics: NAACL 2022, pages 1455–1468, Seattle, United States. Association for Computational Linguistics.

Pablo Villalobos, Anson Ho, Jaime Sevilla, Tamay Besiroglu, Lennart Heim, and Marius Hobbhahn. 2022. Will we run out of data? limits of llm scaling based on human-generated data. arXiv preprint arXiv:2211.04325.

David Wan and Mohit Bansal. 2022. FactPEGASUS: Factuality-aware pre-training and fine-tuning for abstractive summarization. In Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, pages 1010–1028, Seattle, United States. Association for Computational Linguistics.

Alex Wang, Richard Yuanzhe Pang, Angelica Chen, Jason Phang, and Samuel R Bowman. 2022a. Squality: Building a long-document summarization dataset the hard way. ArXiv preprint, abs/2205.11465.

Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R Bowman. 2018.

Glue: A multi-task benchmark and analysis platform for natural language understanding. arXiv preprint arXiv:1804.07461.

Junxiong Wang, Tushaar Gangavarapu, Jing Nathan Yan, and Alexander M Rush. 2024. Mambabyte: Token-free selective state space model. In First Conference on Language Modeling.

Yizhong Wang, Yeganeh Kordi, Swaroop Mishra, Alisa Liu, Noah A Smith, Daniel Khashabi, and Hannaneh Hajishirzi. 2022b. Self-instruct: Aligning language model with self generated instructions. arXiv preprint arXiv:2212.10560.

Yizhong Wang, Swaroop Mishra, Pegah Alipoormolabashi, Yeganeh Kordi, Amirreza Mirzaei, Atharva Naik, Arjun Ashok, Arut Selvan Dhanasekaran, Anjana Arunkumar, David Stap, et al. 2022c. Super-naturalinstructions: Generalization via declarative instructions on 1600+ nlp tasks. In Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing, pages 5085–5109.

Jason Wei, Maarten Bosma, Vincent Zhao, Kelvin Guu, Adams Wei Yu, Brian Lester, Nan Du, Andrew M Dai, and Quoc V Le. 2021. Finetuned language models are zero-shot learners. In International Conference on Learning Representations.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed H Chi, Quoc V Le, Denny Zhou, et al. 2022. Chain-of-thought prompting elicits reasoning in large language models. In Advances in Neural Information Processing Systems.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Remi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander Rush. 2020. Transformers: State-of-the-art natural language processing. In Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations, pages 38–45, Online. Association for Computational Linguistics.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, et al. 2019. Huggingface's transformers: State-of-the-art natural language processing. arXiv preprint arXiv:1910.03771.

Mitchell Wortsman, Tim Dettmers, Luke Zettlemoyer, Ari Morcos, Ali Farhadi, and Ludwig Schmidt. 2023. Stable and low-precision training for large-scale vision-language models. arXiv preprint arXiv:2304.13013.

Guangxuan Xiao, Ji Lin, Mickael Seznec, Julien Demouth, and Song Han. 2022a. Smoothquant: Accurate and efficient post-training quantization for large language models. arXiv preprint arXiv:2211.10438.

Wen Xiao, Iz Beltagy, Giuseppe Carenini, and Arman Cohan. 2022b. PRIMERA: Pyramid-based masked sentence pre-training for multi-document summarization. In Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), pages 5245–5263, Dublin, Ireland. Association for Computational Linguistics.

Tianbao Xie, Chen Henry Wu, Peng Shi, Ruiqi Zhong, Torsten Scholak, Michihiro Yasunaga, Chien-Sheng Wu, Ming Zhong, Pengcheng Yin, Sida I Wang, et al. 2022. Unifiedskg: Unifying and multi-tasking structured knowledge grounding with text-to-text language models. arXiv preprint arXiv:2201.05966.

Wenhan Xiong, Anchit Gupta, Shubham Toshniwal, Yashar Mehdad, and Scott Yih. 2023. Adapting pre-trained text-to-text models for long text sequences. In Findings of the Association for Computational Linguistics: EMNLP 2023, pages 5566–5578, Singapore. Association for Computational Linguistics.

Wenhan Xiong, Jingyu Liu, Igor Molybog, Hejia Zhang, Prajjwal Bhargava, Rui Hou, Louis Martin, Rashi Rungta, Karthik Abinav Sankararaman, Barlas Oguz, Madian Khabsa, Han Fang, Yashar Mehdad, Sharan Narang, Kshitiz Malik, Angela Fan, Shruti Bhosale, Sergey Edunov, Mike Lewis, Sinong Wang, and Hao Ma. 2024. Effective long-context scaling of foundation models. In Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers), pages 4643–4663, Mexico City, Mexico. Association for Computational Linguistics.

Linting Xue, Aditya Barua, Noah Constant, Rami Al-Rfou, Sharan Narang, Mihir Kale, Adam Roberts, and Colin Raffel. 2022. ByT5: Towards a token-free future with pre-trained byte-to-byte models. Transactions of the Association for Computational Linguistics, 10:291–306.

Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William Cohen, Ruslan Salakhutdinov, and Christopher D Manning. 2018. Hotpotqa: A dataset for diverse, explainable multi-hop question answering. In Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing, pages 2369–2380.

Zhewei Yao, Reza Yazdani Aminabadi, Minjia Zhang, Xiaoxia Wu, Conglong Li, and Yuxiong He. 2022. Zeroquant: Efficient and affordable post-training quantization for large-scale transformers. arXiv preprint arXiv:2206.01861.

Lili Yu, Dániel Simig, Colin Flaherty, Armen Aghajanyan, Luke Zettlemoyer, and Mike Lewis. 2023. Megabyte: Predicting million-byte sequences with multiscale transformers. Advances in Neural Information Processing Systems, 36:78808–78823.

Elad Ben Zaken, Shauli Ravfogel, and Yoav Goldberg. 2021. Bitfit: Simple parameter-efficient fine-tuning for transformer-based masked language-models. arXiv preprint arXiv:2106.10199.

Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. 2019. HellaSwag: Can a machine really finish your sentence? In Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics, pages 4791–4800, Florence, Italy. Association for Computational Linguistics.

Aohan Zeng, Xiao Liu, Zhengxiao Du, Zihan Wang, Hanyu Lai, Ming Ding, Zhuoyi Yang, Yifan Xu, Wendi Zheng, Xiao Xia, et al. 2022. Glm-130b: An open bilingual pre-trained model. arXiv preprint arXiv:2210.02414.

Biao Zhang and Rico Sennrich. 2019. Root mean square layer normalization. In Advances in Neural Information Processing Systems, volume 32. Curran Associates, Inc.

Jingqing Zhang, Yao Zhao, Mohammad Saleh, and Peter J. Liu. 2020a. PEGASUS: pre-training with extracted gap-sentences for abstractive summarization. In Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event, volume 119 of Proceedings of Machine Learning Research, pages 11328–11339. PMLR.

Susan Zhang, Stephen Roller, Naman Goyal, Mikel Artetxe, Moya Chen, Shuohui Chen, Christopher Dewan, Mona Diab, Xian Li, Xi Victoria Lin, et al. 2022. Opt: Open pre-trained transformer language models. arXiv preprint arXiv:2205.01068.

Tianyi Zhang, Varsha Kishore, Felix Wu, Kilian Q. Weinberger, and Yoav Artzi. 2020b. Bertscore: Evaluating text generation with BERT. In 8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020. OpenReview.net.

Ming Zhong, Da Yin, Tao Yu, Ahmad Zaidi, Mutethia Mutuma, Rahul Jha, Ahmed Hassan Awadallah, Asli Celikyilmaz, Yang Liu, Xipeng Qiu, and Dragomir Radev. 2021a. QMSum: A new benchmark for query-based multi-domain meeting summarization. In Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, pages 5905–5921, Online. Association for Computational Linguistics.

Ruiqi Zhong, Kristy Lee, Zheng Zhang, and Dan Klein. 2021b. Adapting language models for zero-shot learning by meta-tuning on dataset and prompt collections. arXiv preprint arXiv:2104.04670.

Chapter 6

Appendix BLT

6.1 Model Hyper Parameters

Table 6.1 shows different hyper parameter settings for BLT models.

	Encoder				Global Latent Transf.				Decoder				Cross-Attn.	
Model	$l_{\mathcal{E}}$	#heads	$h_{\mathcal{E}}$	#Params	$l_{\mathcal{G}}$	#heads	$h_{\mathcal{G}}$	#Params	$l_{\mathcal{D}}$	#heads	$h_{\mathcal{D}}$	#Params	#heads	k
400M	1	12	768	7M	24	10	1280	470M	7	12	768	50M	10	2
1B	1	16	1024	12M	25	16	2048	1B	9	16	1024	113M	16	2
2B	1	16	1024	12M	26	20	2560	2B	9	16	1024	113M	16	3
4B	1	16	1024	12M	36	24	3072	4.1B	9	16	1024	113M	16	3
8B	1	20	1280	20M	32	32	4096	6.4B	6	20	1280	120M	20	4

Table 6.1: Architectural hyper-parameters for different BLT model sizes that we train for FLOP-controlled experiments described in this paper.

6.2 FLOPs Equations

Here, we provide the equations used for FLOP computation for the forward-pass of transformer and BLT models based on Hoffmann et al. [2022]; Kaplan et al. [2020]; Casson [2023]. We assume that the backward pass uses twice as much FLOPs as the forward pass.

Operation	FLOPs per token/byte
Attention (l, h_k, n_{heads}, m)	$4 \times l \times h_k \times n_{heads} \times \frac{m+1}{2}$
QKVO (l, h, r)	$(r \times 2 + 2) \times 2 \times l \times h^2$
Feed-forward (l, h, d_{ff})	$2 \times l \times 2 \times h \times d_{ff}h$
De-Embedding (h, V)	$2 \times h \times \|V\|$
Cross-Attention $(l, h_k, n_{heads}, p, r)$	Attention(l, h_k, n_{heads}, p) + QKVO$(l, h_k \times n_{heads}, r)$

Table 6.2: FLOPs for operations used in transformer and BLT models. l corresponds to layers, h is the hidden dimension (h_k with n_{heads} heads), m is the context length, $d_{ff} = 4$ is the feed-forward dimension multiplier, p is the patch size, and r is the ratio of queries to keys.

For a transformer model with l layers, hidden dimension h, context length m, n_{heads} attention heads of dimension h_k, and a feed-forward multipler of d_{ff}, we compute FLOPs as:

$$\begin{aligned} \text{Transformer-FLOPs}(l, h, m, n_{heads}, h_k, d_{ff}, V) &= \text{Feed-forward}(l, h, d_{ff}) && (6.1) \\ &+ \text{QKVO}(l, h, r = 1) && (6.2) \\ &+ \text{Attention}(l, h_k, n_{heads}, m) && (6.3) \\ &+ \text{De-Embedding}(h, V) && (6.4) \end{aligned}$$

For BLT models, we use the above-mentioned primitives together with the FLOPs equation from Section 2.4.5 to compute total FLOPs.

6.3 Rolling Polynomial Hashing

Given a byte n-gram $g_{i,n} = \{b_{i-n+1}, \ldots, b_i\}$, the rolling polynomial hash of $g_{i,n}$ is defined as:

$$\text{Hash}(g_{i,n}) = \sum_{j=1}^{n} b_{i-j+1} a^{j-1} \tag{6.5}$$

Where a is chosen to be a 10-digit prime number.

6.4 Frequency-based n-gram Embedddings

Prior to using hash n-gram embeddings in the final BLT architecture, we also experimented with frequency-based n-gram embeddings. For each $n \in \{1, 2, 3, 4, 5, 6, 7, 8\}$ there is an embedding matrix E_n^{ngram} that contains the most frequent byte-grams for the given n. Since it is intractable to store embeddings as n grows, we only store embeddings for the most frequent $100,000$ byte-grams for each byte-gram. If a particular position i includes an n-gram present in the corresponding the embedding matrix, then this embedding is passed to the next step, encoder multi-headed cross-attention. If a byte-gram is infrequent and therefore not in the matrix, then its embedding is obtained from encoder hash embeddings instead.

Since frequency-based n-grams are limited by the vocabulary of the n-gram tables with infrequent n-grams not being represented at all, we subsequently moved to hash-based n-gram embeddings. See Table 6.3 for a comparison of hash and frequency based n-gram embeddings.

					bpb			
Hash Ngram Sizes	Per Hash Ngram Vocab	Ngram Sizes	Per Ngram Vocab	Total Vocab	Wikipedia	CC	Github	Train Dist
-	-	-	-	-	0.892	0.867	0.506	0.850
6,7,8	50k	6,7,8	50k	300k	0.878	0.860	0.497	0.843
6,7,8	100k	-	-	300k	0.873	0.860	0.499	0.842
6,7,8	100k	6,7,8	100k	600k	0.868	0.857	0.494	0.839
6,7,8	200k	-	-	600k	0.862	0.856	0.492	0.838
3,4,5	50k	3,4,5	50k	300k	0.862	0.856	0.491	0.837
3,4,5	100k	-	-	300k	0.859	0.855	0.491	0.837
6,7,8	200k	6,7,8	200k	1M	0.861	0.855	0.491	0.837
6,7,8	400k	-	-	1M	0.855	0.853	0.491	0.834
3,4,5,6,7,8	50k	3,4,5,6,7,8	50k	600k	0.855	0.853	0.488	0.834
3,4,5	100k	3,4,5	100k	600k	0.851	0.853	0.486	0.834
3,4,5	200k	-	-	600k	0.850	0.852	0.485	0.833
3,4,5,6,7,8	100k	-	-	600k	0.850	0.852	0.486	0.833
3,4,5	400k	-	-	1M	0.844	0.851	0.483	0.832
3,4,5	200k	3,4,5	200k	1M	0.843	0.850	0.482	0.830
3,4,5,6,7,8	100k	3,4,5,6,7,8	100k	1M	0.844	0.850	0.482	0.830
3,4,5,6,7,8	200k	-	-	1M	0.840	0.849	0.481	0.830
3,4,5,6,7,8	200k	3,4,5,6,7,8	200k	2M	**0.833**	**0.846**	**0.478**	**0.826**
3,4,5,6,7,8	400k	-	-	2M	**0.831**	**0.846**	**0.478**	**0.826**

Table 6.3: Ablations on the use of frequency-based as well as hash-based n-gram embedding tables for a 1B BLT model trained on 100B bytes.

6.5 Entropy Patching Example from MMLU

We illustrate how a few-shot example from a downstream task i.e. MMLU [Hendrycks et al., 2021], is patched using an entropy-model trained for use with BLT models in Figure 6.1. Directly using the entropy model with the full-context window causes repetitive patterns to be heavily patched. For example, "10 times, with an rms deviation of about" in the MMLU query is patched frequently the first time it is encountered, but is part of very large patches the next three times, which, although inference efficient, maybe undesirable for reasoning. One method that we use to avoid such a "entropy" drift is by resetting the entropy context with new lines and using a approximate monotonicity constraint (see Section 2.4.4).

Figure 6.1: An example of default entropy-based patching with global threshold during inference on MMLU. Green denotes the prompt, Blue denotes the few-shot examples, and red denotes the question to be answered. Note that the size of the patches for the repeated phrases in the answer choices is much larger, which means that the global model is invoked significantly fewer times than its tokenizer-based counterpart, with this inference patching scheme.

Chapter 7

Appendix Socratic Pretraining

7.1 Appendix Socratic Pretraining

7.1.1 Dataset Information

We use the QMSum and SQuALITY datasets according to their intended research purposes.

Dataset	Domain	# Ex.	Doc. Len	Sum. Len
CNN/DM	news	311K	804	60
XSum	news	226K	438	24
QMSum	meetings	1,808	9,067	70
SQuALITY	stories	625	5,200	237

Table 7.1: Statistics of general summarization vs. QFS datasets, length in words [Wang et al., 2022a].

7.1.2 Training Details

We describe here the training details for SOCRATIC pretraining as well as downstream task finetuning. Our experiments rely on the Huggingface Transformers library [Wolf et al., 2020]. Our code includes sample pretraining and finetuning scripts to facilitate the reproduction of our results. We use 8 Nvidia A100 GPUs to run the experiments described in this paper. We will release our code under BSD 3-Clause license.

Pretraining

Data Preprocessing In the Books3 corpus, documents are longer than the desired input and target texts, we therefore segment the documents to obtain roughly the desired lengths. In the UnDial dataset, the opposite is true and therefore we concatenate dialogues to obtain the desired lengths. Following this segmentation or concatenation, we mask the input text and construct the target as described in section 3.3 depending on the desired mode. We then truncate the input and target texts to 256 and 512 tokens respectively.

Special Tokens We introduce mode tokens and a new separator tokens to the tokenizer of the BART-large model before the pretraining adaptation step.

Training Hyperparameters We train the BART-large model for 100k steps with batch size 512, checkpointing every 10k steps. For the ablations, we use batch size of 64 and the same number of steps. In all our experiments, we use AdamW optimizer with 5k warmup steps, learning rate 3e-5, weight decay of 0.01, max grad norm of 0.1, and bfloat16. Our choice of hyperparameters is based on best practices from previous work performing pretraining adaptations of BART-large [Xiao et al., 2022b; Wan and Bansal, 2022]. We also performed grid-search on the learning rate on the small-scale pretraining dataset testing the values {3e-6, 3e-5, 1e-4} but finding the initial value to perform best. We use the same hyperparameters on all three pretraining corpora in our ablations.

Checkpoint validation We evaluate the checkpoints on the validation dataset of the target downstream tasks and pick the best performing checkpoint.

Finetuning

SegEnc Implementation We use the SegEnc implementation from the original authors. Instead of using vanilla BART-large to initialize the SegEnc model, we use one of our pretrainined models.

Finetuning Hyperparameters We use the same hyperparameters for both QMSum and SQuALITY datasets and for QFS and finegrained planning experiments. We train the SegEnc model for 10 epochs with batch size 1 and bfloat16. We use the AdamW optimizer with learning rate 5e-6. We tested the following learning

rate values {5e-7, 5e-6, 5e-5, 5e-4}. We use beam search decoding with beam size of 4. Our hyperparameters follow the best performing hyperparameters found by the original authors of the SegEnc model [Vig et al., 2022]. Annotations will be made available ensuring the identity of the workers remains anonymous. We will only report the answers to the questions for each example and anonymize the worker ID.

Mode While the SOCRATIC pretraining consists of both Reconstruct and Ask&Answer modes, we found that the latter performed best on the downstream tasks.

7.1.3 Automated Evaluation Details

We perform an automated evaluation using Rouge and BERTScore metrics following best practices from previous work. Specifically, we use the evaluation setup from Vig et al. [2022] for QMSum and the evaluation setup from Wang et al. [2022a] for SQuALITY. More details and the relevant scripts can be found the in the supporting code supporting their papers. We also provide scripts to reproduce our evaluation. For BERTScore, we report recall following recommendations from Liu et al. [2022b].

7.1.4 Human Evaluation Details

We perform a human evaluation study to confirm that variations between models are perceptible and meaningful to human users. The study separately assesses the QFS and the finegrained planning models finetuned on the QMSum dataset. In both cases, we use 100 of the 281 examples from the QMSum test set, and three independent annotators from the Amazon Mechanical Turk platform. We restrict the study to the specific questions of the QMSum dataset as these also provide relevant text spans in the original dialogue.

We measure inter-annotator agreement with Fleiss Kappa κ [Fleiss, 1971] and obtain fair to moderate agreement in our tasks. Other studies that also rely on untrained crowd-sourced workers report similar, or sometimes even lower, agreement [Goyal et al., 2022b].

QFS Task In this task, we compare the SegEnc model with SOCRATIC pretraining to Pegasus and BART pretraining. We ask annotators to select the best answer to the given query between two candidate summaries or mark if they are equally good. We provide both the reference summary and the relevant text span as

supporting information. Annotator agreement on this task is $\kappa = 0.33$. The results are summarized in Figure 3.6 and the annotation instructions can be found in Figure 7.1.

Finegrained Planning Task In this task, we compare the SOCRATIC SegEnc model to the baseline BART SegEnc model in terms of their adherence to a finegrained plan. Both models are finetuned to the finegrained planning task on QMSum with the content question control strategy. Here we test how well they follow oracle plans automatically generated from the reference summary. The task is structured in two parts. First, for each question of the oracle plan, we ask annotators whether a sentence of the summary answers the question. We repeat for both SOCRATIC and BART summaries. On this task, we obtain moderate agreement of $\kappa = 0.49$. Next, we ask the annotators to select the best summary between the two candidates in terms of how closely it follows the plan. For the second task, the agreement is $\kappa = 0.34$. The results are summarized in Figure 3.9 and the annotation instructions can be found in Figure 7.2.

Worker Selection and Considerations An ethics review board did not review this particular protocol, but we followed prior protocols and internally-approved guidelines, such as carefully calibrating the time/HIT to ensure a pay-rate of $12/hour and letting workers know that their annotations will be used as part of a research project to evaluate the performance of summarization systems.

We selected workers according to the following criteria: HIT approval rate greater than or equal to 98%, number of HITs approved greater than or equal to 10000, and located in either the United Kingdom or the United States. The workers also passed a qualification test for a related summarization task from a prior project, ensuring that the annotators were familiar with the task of judging model-generated summaries.

7.1.5 Comparing Control Strategies

Using content questions for QG augmentation in SOCRATIC pretraining improves performance across control strategies, including on non-question-based finegrained controls like keyword chains (see Table 3.2). While most previous work has focused on keyword controls [He et al., 2022a] and fact-oriented questions for text generation [Narayan et al., 2022], there are inherent limitations with these approaches. We identify important qualitative properties of queries for controllable generation below that informed our choice of content questions for SOCRATIC pretraining.

Control Type	Length	Lexical Overlap With Summ.	Lexical Overlap Across Queries	
	% of summ. len	Rouge 1	Avg. Overlap	Max. Overlap
Keywords	25%	37.9	43%	100%
Bleuprint QA	149%	65.9	22%	44%
Content Questions (ours)	48%	38.1	36%	67%

Table 7.2: Properties of finegrained control strategies for the QMSum dataset. We measure lexical overall between the control sequence and the reference summary. We also calculate the average and maximum lexical overlap of two control sequences from the same QMSum document but answering two different high-level queries.

Natural To facilitate the use of controllable summarization, one overarching objective is to make the user interaction with the system as natural as possible. When evaluating how "natural" a query strategy is, we consider whether such a strategy is used by humans when they interact with one another. According to this perspective, using keywords is an unnatural query strategy. Users generally express themselves through natural language, and when inquiring about information, they use questions. Our query systems in controllable summarization should strive to reflect this and support natural queries from the users.

Unambiguous To ensure that summaries contain the intended information, it is necessary that queries refer with minimal ambiguity to the information of interest in the document. When dealing with long documents, where the same entities occur repeatedly, keywords often imprecisely describe the intended query. But it is precisely with such long documents that query-focused summarization is particularly useful. In Table 7.2, we show that different keyword queries about the same document have a lexical overlap of 46% of words on average and 100% in the worst-case scenario in QMSum. In comparison, content questions have a word overlap of 36% on average and no more than 67%. When formulating queries in natural language, they more richly encode the entities and their relations making them less ambiguous.

Concise Fact-oriented question-answer pairs (blueprint QA) [Narayan et al., 2022] tend to be less ambiguous than keywords (with the least lexical overlap across the three query strategies) but often end up requiring more text than the summary itself. On average, blueprint QA uses 50% more words than the summary (see Table 7.2). This makes this query strategy impractical for controllable summarization where the concision of the query is a desirable property.

Instructions

In this task, you will compare two candidate summaries and pick the one that provides the most *informative and correct* answer to the given question.
To correctly solve this task, follow these steps:

1. Carefully read the question, reference summary, and candidate summaries. If needed refer to the dialogue.
2. Compare the two candidate summaries according to the following criteria:
    a. *Informativeness of the summary.* How much information does the answer contain? The more informative the better.
    b. *Relevance of the information.* How relevant to the question is the information in the answer? The more relevant the information the better.
    c. *Correctness of the information.* How correct is the information in the answer? The fewer errors in the answer the better.
3. **Pick the summary that provides the most informative and correct answer to the given question.**

Note that the reference summary is provided as a guide for a satisfactory answer and that candidate summaries might be more informative than the reference.

Examples

Reference: Product Manager said that it is important to think of the weight of the device. User Interface agreed and mentioned that the current design was too heavy compared to the competition. Product manager concluded that reducing the weight of the device should be the focus of the team's work for the following week.

Example 1

Question: What what said about the weight of the device?
Summary 1: Product manager said weight is important. User Interface agreed the team should focus on reducing the device's weight.
Summary 2: Product manager noted that the device felt heavy and that the team might have overlooked this aspect. User interface mentioned the current design was heavier than the competition. Product manager concluded that the team should focus on a lighter design and present it during next week's meeting.
Explanation: Summary 2 is correct. Summary 2 is more informative than Summary 1. Summary 1 also contains a mistake: It was Product Manager that told the team to focus on reducing the weight of the device, not User Interface.

Warning: Annotations will be checked for quality against control labels, **low quality work will be rejected.**

Figure 7.1: QFS human annotation instructions.

Instructions

Part 1

In this task, you will be given a list of questions and a summary. For each question, you will **determine whether it is correctly answered by one of the sentences of the summary**. Each sentence of the summary can only correspond to **one** specific question. **If the question says BLANK, please mark "Not Answered."**
You will judge two summaries in this fashon.

Part 2

After judging two summaries in Part 1, you will **pick the summary that most closely answers the list of questions**. You will pick the best summary based on the number of questions it correctly answers and if the answers appear in the same order as the questions. We do provide an option to rate the summaries as equal, but note that **this is very rare/may not occur**.

Example

Part 1

Task: Which one of the specific questions is answered by a sentence of the summary?
Summary 1: Product manager noted that the device felt heavy and that the team might have overlooked this aspect. User interface mentioned the current design was heavier than the competition. Product manager concluded that the team should focus on a lighter design and present it during next week's meeting.
Questions:

- Question 1: What did Product manager note about the device?
- Question 2: What did User interface mention about the competition?
- Question 3: What was the conclusion?

Explanation:
Each one of the three questions is answered by a different sentence of summary 1.

Task: Which one of the specific questions is answered by a sentence of the summary?
Summary 2: Product manager noted that the device felt heavy and that the team might have overlooked this aspect. Product manager concluded that the team should focus on a lighter design and present it during next week's meeting.
Questions:

- Question 1: What did Product manager note about the device?
- Question 2: What did User interface mention about the competition?
- Question 3: What was the conclusion?

Explanation:
Question 1 and 3 are answered by summary 2. Question 2 was not answered by any sentence in the summary 2.

Part 2

Task: Which summary better answers the following questions in their order?
Summary 1: Product manager noted that the device felt heavy and that the team might have overlooked this aspect. User interface mentioned the current design was heavier than the competition. Product manager concluded that the team should focus on a lighter design and present it during next week's meeting.
Summary 2: Product manager noted that the device felt heavy and that the team might have overlooked this aspect. Product manager concluded that the team should focus on a lighter design and present it during next week's meeting.
Questions:

- Question 1: What did Product manager note about the device?
- Question 2: What did User interface mention about the competition?
- Question 3: What was the conclusion?

Explanation:
Summary 1. Summary 1 answers 3/3 of the questions while Summary 2 only 2/3.

Warning: Annotations will be checked for quality against control labels, **low quality work will be rejected.**

Figure 7.2: Finegrained planning human annotation instructions.

Chapter 8

Appendix QLoRA

8.1 Inference Speedups for NF4

The speedups for single batch inference for NF4 compare to 16-bit float are shown in Figure 8.1.

8.2 Environmental Impact and Carbon Footprint Reductions

The relationship between serving efficiency and environmental impact and cost/performance is a complicated one. The short answer is, for personal use, we reduce the environmental impact per token by about 3.5x. If 50% LLM deployments are personal and 50% company, we reduce the environmental footprint of inference by 72%.

The long answer is this: For deployment, there are two options: (1) personal deployment, (2) deployment by companies for many users. (1) uses small batch sizes (usually batch size =1), (2) uses large batch sizes (64-128). Per token, (2) offers about >50x better efficiency because for every weight that is loaded from memory, up to 64-128 tokens can be calculated. The >50x improvement in efficiency stems from the fact memory operations are energy inefficient, and floating point operations are energy efficient. As such,company deployment (2) is a very environmentally friendly and cheap approach.

Currently, we have efficient 4-bit CUDA kernels for the personal deployment scenario (1) with batch size=1 which are about 3.5x more efficient (see Figure 8.1). As such, we reduce the overall footprint for personal deployment (1) significantly.

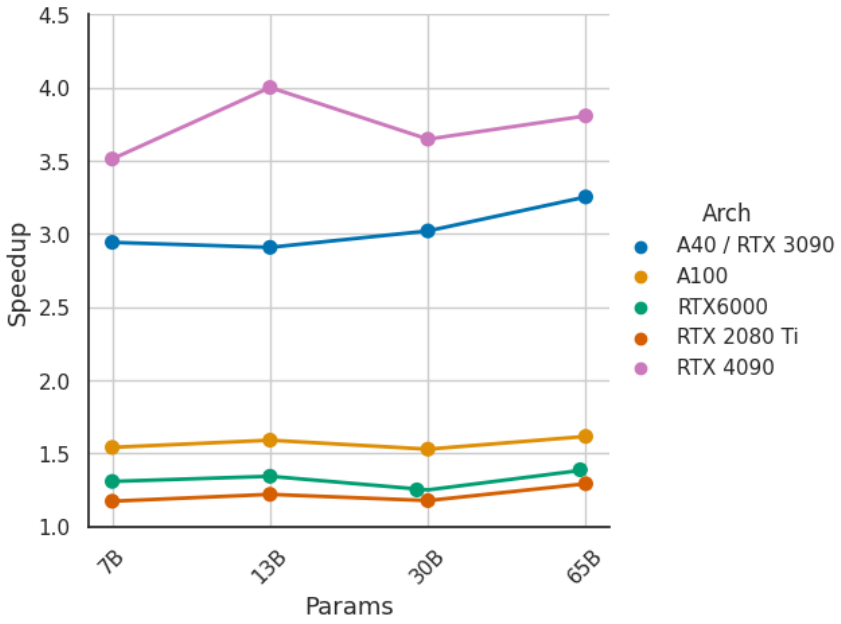

Figure 8.1: Speedups of NF4 inference for batch size 1 compared to 16-bit inference for different GPUs. We see that RTX 3090/4090 and A40 GPUs have large speedups of 2.9-4.0x while other GPUs have speedups in the range 1.1-1.5x. The difference between GPUs is mostly caused by poor instruction throughput bottlenecks.

The overall cost and footprint is now determined by how many people use (1) vs (2). For example, if 50% of users use personal deployment (1) and 50% company deployments (2) and we assume that (2) is about 50x more efficient then we get the following numbers. Approach (1) accounts for 50%/(50% + 50%/50) = 98% of environmental impact. This means, the more users deploy personal LLMs, the larger benefit of our method. If 50% of people use personally deployed LLMs (phones, laptops etc) then our method will reduce the impact by about 1- (98%/3.5x) = 72%.

The environmental impact reduction is roughly proportional to energy consumption per token processed which is roughly proportional to the cost of running LLMs. As such, the bottom line is affected in the same proportions.

Overall, our method will have a strong impact on environmental impact and cost reduction for personal LLM deployments.

8.3 QLoRA vs Standard Finetuning Experimental Setup Details

8.3.1 BF16 vs NF4 for T5/RoBERTa

Here we detail additional results where we run experiments to confirm we can replicate 16-bit full finetuning performance when we use QLoRA. We test RoBERTA and T5 models sized 125M to 3B parameters on GLUE and the Super-NaturalInstructions dataset. Results are shown in Table 8.1. In both datasets, we observe that 16-bit, 8-bit, and 4-bit adapter methods replicate the performance of the fully finetuned 16-bit baseline. This suggests that the performance lost due to the imprecise quantization can be fully recovered through adapter finetuning after quantization.

8.3.2 Hyperparameter search for QLoRA

We do a hyperparameter search for LoRA over the following variables: LoRA dropout { 0.0, 0.05, 0.1}, LoRA r { 8, 16, 32, 64, 128, 256}, LoRA layers {key+query, all attention layers, all FFN layers, all layers, attention + FFN output layers}. We keep LoRA α fixed and search the learning rate, since LoRA α is always proportional to the learning rate.

We find that LoRA dropout 0.05 is useful for small models (7B, 13B), but not for larger models (33B,

Dataset	GLUE (Acc.)	Super-NaturalInstructions (RougeL)				
Model	RoBERTa-large	T5-80M	T5-250M	T5-780M	T5-3B	T5-11B
BF16	88.6	40.1	42.1	48.0	54.3	62.0
BF16 replication	88.6	40.0	42.2	47.3	54.9	-
LoRA BF16	88.8	40.5	42.6	47.1	55.4	60.7
QLoRA Int8	88.8	40.4	42.9	45.4	56.5	60.7
QLoRA FP4	88.6	40.3	42.4	47.5	55.6	60.9
QLoRA NF4 + DQ	-	40.4	42.7	47.7	55.3	60.9

Table 8.1: Experiments comparing 16-bit BrainFloat (BF16), 8-bit Integer (Int8), 4-bit Float (FP4), and 4-bit NormalFloat (NF4) on GLUE and Super-NaturalInstructions. QLoRA replicates 16-bit LoRA and full-finetuning.

65B). We find LoRA r is unrelated to final performance if LoRA is used on all layers as can be seen in Figure 8.2

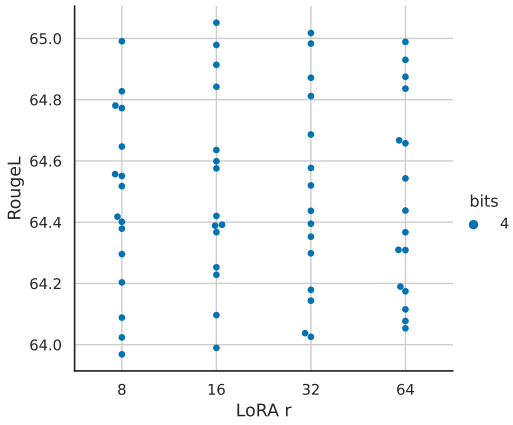

Figure 8.2: LoRA r for LLaMA 7B models finetuned on Alpaca. Each dot represents a combination of hyperparameters and for each LoRA r we run 3 random seed with each hyperparameter combination. The performance of specific LoRA r values appears to be independent of other hyperparameters.

8.3.3 Super-Natural Instructions Experimental Setup Details

We use the same preprocessing of the Super-Natural Instruction dataset as **?**. However, we split the training data in training and validation datasets allowing us to perform more rigorous hyperparameter tuning and early stopping. We use the same hyperparameters described in the paper for training the various T5 model sizes on the Super-Natural Instruction data. We use LoRA $r = 16$ for small, medium, and large T5 models and LoRA $r = 64$ for T5 xl and xxl models. We also use LoRA $\alpha = 64$ in all our experiments and no LoRA

dropout.

8.4 Training a State-of-the-art Chatbot Experimental Setup Details

8.4.1 Datasets

We describe the datasets used for QLoRA finetuning experiments outlined in Section 4.5.

OASST1 The OpenAssistant dataset [Köpf et al., 2023] was collected via crowd-sourcing. It contains 161,443 unique messages distributed across 66,497 conversations and spanning 35 different languages. The dataset often contains several ranked replies for each given user question. In our experiments, we only use the top reply at each level in the conversation tree. This limits the dataset to 9,846 examples. We finetune models on the full conversation including the user queries.

HH-RLHF This is a human preference dataset about helpfulness and harmlessness. Each datapoint consists of two assistant replies to a user question along with a human preference judgment of the best reply. The dataset contains 160,800 examples. When finetuning on this dataset, we combine helpfulness and harmlessness data and only keep the preferred assistant reply.

FLAN v2 The FLAN v2 collection [Longpre et al., 2023] is a collection of 1836 tasks augmented with hundreds of manually curated templates and rich formatting patterns into over 15M examples. The authors show that models trained on this collection outperform other public collections including the original FLAN 2021 [Wei et al., 2021], T0++ [Sanh et al., 2021], Super-Natural Instructions [**?**], and OPT-IML [Iyer et al., 2022]. We used the same task mixtures described by the authors with the exception of some datasets that were not freely available at the time of writing.

Self-Instruct, Alpaca, Unnatural Instructions The Self-Instruct, Alpaca, and Unnatural Instructions datasets [Wang et al., 2022b; Taori et al., 2023; Honovich et al., 2022] are instruction tuning datasets collected with various approaches of model distillation from GPT-3 Instruct and ChatGPT. They rely on prompting, in-context learning, and paraphrasing to come up with diverse sets of instructions and outputs.

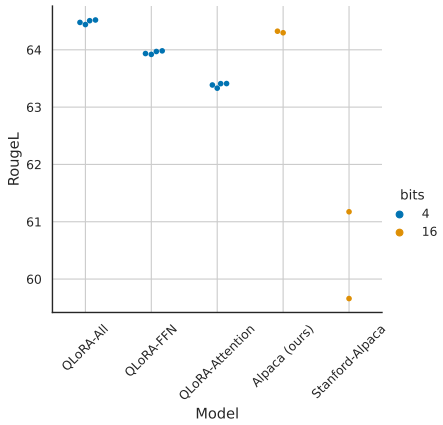

Figure 8.3: RougeL for LLaMA 7B models on the Alpaca dataset. Each point represents a run with a different random seed. We improve on the Stanford Alpaca fully finetuned default hyperparameters to construct a strong 16-bit baseline for comparisons. Using LoRA on all transformer layers is critical to match 16-bit performance.

The datasets comprise of 82,612, 51,942, and 240,670 examples respectively. One advantage of such distilled datasets is that they contain a more diverse set of instruction styles compared to the FLAN v2 collection and similar instruction tuning collections.

Longform The LongForm dataset [Köksal et al., 2023] is based on an English corpus augmented with instructions and as such is a hybrid human-generated dataset. The underlying documents are human-written and come from C4 and Wikipedia while the instructions are generated via LLMs. The dataset is extended with additional structured corpora examples such as Stack Exchange and WikiHow and task examples such as question answering, email writing, grammar error correction, story/poem generation, and text summarization. The dataset contains 23,700 examples.

Chip2 is part of the OIG Laion dataset. It contains Python code examples, natural instruction examples, generic harmless instructions, instruction/responses with lists, follow-up questions, Wikipedia toxic adversarial questions, grade school math, reasoning instructions, and character and scene descriptions with a total of 210,289 examples.

8.4.2 Default LoRA hyperparameters do not match 16-bit performance

We find that default hyperparameters for fully finetuned baselines are undertuned. We do a hyperparameter search over learning rates 1e-6 to 5e-5 and batch sizes 8 to 128 to find robust baselines. Results for 7B LLaMA finetuning on Alpaca are shown in Figure 8.3.

When using the standard practice of applying LoRA to query and value attention projection matrices [Hu et al., 2021], we are not able to replicate full finetuning performance for large base models. As shown in Figure 8.3 for LLaMA 7B finetuning on Alpaca, we find that the most critical LoRA hyperparameter is how many LoRA adapters are used in total and that LoRA on all linear transformer block layers are required to match full finetuning performance. Other LoRA hyperparameters, such as the projection dimension r, do not affect performance.

8.4.3 Hyperparameters

We provide the exact hyperparameters used in our QLoRA finetuning experiments. We find hyperparameters to be largely robust across datasets. We use the MMLU 5-shot dev set for validation and hyperparameter tuning. In all our experiments we use NF4 with double quantization and bf16 computation datatype. We set LoRA $r = 64$, $\alpha = 16$, and add LoRA modules on all linear layers of the base model. We also use Adam beta2 of 0.999, max grad norm of 0.3 and LoRA dropout of 0.1 for models up to 13B and 0.05 for 33B and 65B models. Following previous work on instruction finetuning [Wei et al., 2021; **?**] and after benchmarking other linear and cosine schedules, we use a constant learning rate schedule. We use group-by-length to group examples of similar lengths in the same batch (note this will produce an oscillating loss curve). The hyperparameters we tune for each model size are shown in Table 8.2.

8.4.4 Ablations

While it is general practice in the literature to only train on the response in instruction following datasets, we study the effect of training on the instruction in addition to the response in Table 8.3. In these experiments, we restrict the training data to 52,000 examples and use the 7B model. Over four different instruction tuning datasets, we find that only training on the target is beneficial to MMLU performance. We did not evaluate the effect this may have on chatabot performance as measured by vicuna or OA benchmarks.

Parameters	Dataset	Batch size	LR	Steps	Source Length	Target Length
7B	All	16	2e-4	10000	384	128
7B	OASST1	16	2e-4	1875	-	512
7B	HH-RLHF	16	2e-4	10000	-	768
7B	Longform	16	2e-4	4000	512	1024
13B	All	16	2e-4	10000	384	128
13B	OASST1	16	2e-4	1875	-	512
13B	HH-RLHF	16	2e-4	10000	-	768
13B	Longform	16	2e-4	4000	512	1024
33B	All	32	1e-4	5000	384	128
33B	OASST1	16	1e-4	1875	-	512
33B	HH-RLHF	32	1e-4	5000	-	768
33B	Longform	32	1e-4	2343	512	1024
65B	All	64	1e-4	2500	384	128
65B	OASST1	16	1e-4	1875	-	512
65B	HH-RLHF	64	1e-4	2500	-	768
65B	Longform	32	1e-4	2343	512	1024

Table 8.2: Training hyperparameters for QLoRA finetuning on different datasets and across model sizes.

Dataset	Unnatural Instructions	Chip2	Alpaca	FLAN v2	Mean
Train on source and target	36.2	33.7	38.1	42.0	37.5
Train on target	38.0	34.5	39.0	42.9	38.6

Table 8.3: MMLU 5-shot test results studying the effect of training on the instructions in addition to the response.

8.4.5 Training Considerations

We note that the OASST1 dataset on which BLTmodels are trained is multilingual and that the OA benchmark also contains prompts in different languages. We leave it to future work to investigate the degree to which such multilingual training improves performance on instructions in languages other than English and whether this explains the larger gap between Vicuna-13B model (only trained on English data) and BLT33B and 65B on the OA benchmark.

Given the strong performance of BLTmodels, we investigate any data leakage between the OASST1 data and the Vicuna benchmark prompts. We do not find overlapping prompts after performing fuzzy string matching in the two datasets and inspecting the closest matches manually.

Furthermore, we note that our model is only trained with cross-entropy loss (supervised learning) without relying on reinforcement learning from human feedback (RLHF). This calls for further investigations of the tradeoffs of simple cross-entropy loss and RLHF training. We hope that QLoRA enables such analysis at scale, without the need for overwhelming computational resources.

8.4.6 What is more important: instruction finetuning dataset size or dataset quality?

To understand the effects of dataset quality vs. dataset size, we experiment with subsampling large datasets with at least 150,000 samples (Chip2, FLAN v2, Unnatural Instructions), into datasets of size 50,000, 100,000 and 150,000 and examine the resulting trends, as shown in Table 8.4. We find that increasing the dataset size and increasing the number of epochs improves MMLU only marginally (0.0 - 0.5 MMLU), while the difference between datasets is up to 40x larger (1.5 - 8.0 MMLU). This is a clear indicator that dataset quality rather than dataset size is critical for mean MMLU accuracy. We obtain similar findings for chatbot performance, with the most successful dataset for training, OASST1, containing less than 10k examples after processing.

8.5 Chatbot Evaluation Details

Following common practice, we use the MMLU (Massively Multitask Language Understanding) benchmark [Hendrycks et al., 2020] to measure performance on a range of language understanding tasks. This is

a multiple-choice benchmark covering 57 tasks including elementary mathematics, US history, computer science, law, and more. We report 5-shot test accuracy.

We also test generative language capabilities through both automated and human evaluations. This second set of evaluations relies on queries curated by humans and aims at measuring the quality of model responses. While this is a more realistic testbed for chatbot model performance and is growing in popularity, there is no commonly accepted protocol in the literature. We describe below our proposed setup, using nucleus sampling with $p = 0.9$ and temperature 0.7 in all cases.

8.5.1 Benchmark Data

We evaluate on two curated datasets of queries (questions): the Vicuna prompts [Chiang et al., 2023] and the OASST1 validation dataset [Köpf et al., 2023]. We use the Vicuna prompts, a set of 80 prompts from a diverse set of categories, without modifications. The OASST1 dataset is a multilingual collection of crowd-sourced multiturn dialogs between a user and an assistant. We select all user messages in the validation dataset as queries and include previous turns in the prompt. This procedure leads to 953 unique user queries. We term these two datasets the Vicuna and OA benchmarks.

8.5.2 Automated Evaluation

First, based on the evaluation protocol introduced by Chiang et al. [2023], we use GPT-4 to rate the performance of different systems against ChatGPT (GPT-3.5 Turbo) on the Vicuna benchmark. Given a query along with ChatGPT's and a model's responses, GPT-4 is prompted to assign a score out of ten to both

	Chip			Unnatural Instructions			FLAN v2			
Datapoints ↓ Epochs →	1	2	3	1	2	3	1	2	3	Mean
50000	34.50	35.30	34.70	38.10	42.20	38.10	43.00	43.50	44.10	39.28
100000	33.70	33.90	34.00	40.10	41.20	37.00	43.90	43.70	44.90	39.16
150000	34.40	34.80	35.10	39.70	41.10	41.50	44.60	45.50	43.50	40.02
Mean	34.20	34.67	34.60	39.30	41.50	38.87	43.83	44.23	44.17	

Table 8.4: Effect of different dataset sizes and finetuning epochs on mean 5-shot MMLU test set accuracy. While increasing the dataset size and training for more than 1 epochs helps with MMLU performance, the difference between datasets are far larger, indicating that dataset quality affects MMLU performance more than dataset size.

responses and provide an explanation. The overall performance of a model is calculated as a percentage of the score that ChatGPT achieved. Note this relative score can be higher than 100% if the model achieves a higher absolute score than ChatGPT. We find a significant ordering effect with GPT-4 increasing the score of the response occurring earlier in the prompt. To control for such effects, we recommend reporting the mean score over both orders.

Next, we measure performance through direct comparisons between system outputs. We simplify the rating scheme to a three-class labeling problem that accounts for ties. We prompt GPT-4 to pick the best response or declare a tie and provide an explanation. We conduct these head-to-head comparisons on all permutations of pairs of systems on both the Vicuna and OA benchmarks.

8.5.3 Human Evaluation

While recent work indicates generative models can be effectively employed for system evaluations [Fu et al., 2023], the reliability GPT-4 ratings to assess chatbot performance is, to our knowledge, yet to be proven to correlate with human judgments. Therefore, we run two parallel human evaluations on the Vicuna benchmark matching both automated evaluation protocols described above. We use Amazon Mechanical Turk (AMT) and get two human annotators for comparisons to ChatGPT and three annotators for pairwise comparisons. We conduct a human evaluation with the same wording given to GPT-4 in the original Vicuna evaluation [Chiang et al., 2023], adjusted for an Amazon Mechanical Turk form as show in Figure 8.4.

We report moderate agreement among human annotators (Fleiss $\kappa = 0.42$) with additional deterioration when comparing two strong systems. This points to limitations in the current benchmarks and human evaluation protocols for chatbot task performance. When manually comparing generations from ChatGPT and BLT65B on the Vicuna benchmark, we find that subjective preferences start to play an important role as the authors of this paper disagreed on the many preferred responses. Future work should investigate approaches to mitigate these problems drawing from disciplines that developed mechanisms to deal with subjective preferences, such as Human-Computer Interaction and Psychology.

8.5.4 Elo Rating

With both human and automated pairwise comparisons, we create a tournament-style competition where models compete against each other. The tournament is made up of matches where pairs of models compete to produce the best response for a given prompt. This is similar to how Bai et al. [2022a] and Chiang et al. [2023] compare models, but we also employ GPT-4 ratings in addition to human ratings. We randomly sample from the set of labeled comparisons to compute Elo [Elo, 1967, 1978]. Elo rating, which is widely used in chess and other games, is a measure of the expected win-rate relative to an opponent's win rate, for example, an Elo of 1100 vs 1000 means the Elo 1100 player has an expected win-rate of approximately 65% against the Elo 1000 opponent; a 1000 vs 1000 or 1100 vs 1100 match results in an expected win-rate of 50%. The Elo rating changes after each match proportionally to the expected outcome, that is, an unexpected upset leads to a large change in Elo rating while an expected outcome leads to a small change. Over time, Elo ratings approximately match the skill of each player at playing the game. We start with a score of 1,000 and use $K = 32$. Similar to Chiang et al. [2023], we repeat this procedure 10,000 times with different random seeds to control for ordering effects, e.g., the effect of which model pairs compete with each other first.

8.5.5 Evaluation Biases and Limitations

In our analysis, we also find that automated evaluation systems have noticeable biases. For example, we observe strong order effects with GPT-4 assigning higher scores to the system appearing first in its prompt. The relatively weak sample-level agreement between GPT-4 and human annotators (Fleiss $\kappa = 0.25$) also suggests that human annotators and automated systems might rely on preferences that are not always aligned. In addition, in Table 4.7, we observe that GPT-4 assigns significantly higher scores to its own outputs compared to human ratings, Elo of 1348 vs 1176, which represent an additional 20% probability of winning against an opponent. Future work should examine the presence of potential biases in automated evaluation systems as well as possible mitigation strategies.

While we found that the GPT-4 evaluation gave different results depending on which system was presented first, when averaged over both options the pairwise results were well-ordered. The aggregated pairwise judgments are shown in Table 8.5. On inspection, it is clear these judgments are transitive, i.e., when

Model	Guanaco 65B	Guanaco 33B	Vicuna	ChatGPT-3.5 Turbo	Bard	Guanaco 13B	Guanaco 7B
Guanaco 65B	-	0.21	0.19	0.16	0.72	0.59	0.86
Guanaco 33B	-0.21	-	0.17	0.10	0.51	0.41	0.68
Vicuna	-0.19	-0.17	-	0.10	0.50	0.20	0.57
ChatGPT-3.5 Turbo	-0.16	-0.10	-0.10	-	0.35	0.19	0.40
Bard	-0.72	-0.51	-0.50	-0.35	-	0.12	0.03
Guanaco 13B	-0.59	-0.41	-0.20	-0.19	-0.12	-	0.20
Guanaco 7B	-0.86	-0.68	-0.57	-0.40	-0.03	-0.20	-

Table 8.5: Aggregated pairwise GPT-4 judgments between systems where the value of a cell at row x and column y is $\frac{\text{\# judgment } x \text{ is better than } y - \text{\# judgment } y \text{ is better than } x}{\text{total \# number of judgments}}$

Model	Params	Size
Guanaco	65B	41 GB
Guanaco	33B	21 GB
Vicuna	13B	26 GB
ChatGPT-3.5 Turbo	N/A	N/A
Bard	N/A	N/A
Guanaco	13B	10 GB
Guanaco	7B	5 GB

Table 8.6: The complete ordering induced by pairwise GPT-4 judgments between systems.

System A is judged better than System B and System B is judged better than System C, it is always the case that System A is judged better than System C. This yields a complete ordering, given in Table 8.6.

8.6 NormalFloat 4-bit Data Type

The steps needed to reconstruct the 4-bit Normal Float (NF4) data type are detailed in Figure 8.5

The exact values expressible in the NF4 data type are as follows:

[-1.0, -0.6961928009986877, -0.5250730514526367,
-0.39491748809814453, -0.28444138169288635, -0.18477343022823334,
-0.09105003625154495, 0.0, 0.07958029955625534, 0.16093020141124725,
0.24611230194568634, 0.33791524171829224, 0.44070982933044434,
0.5626170039176941, 0.7229568362236023, 1.0]

8.6.1 Normality of Trained Neural Network Weights

While it is commonly assumed that trained neural network weights are mostly normally distributed, we perform statistical testing to verify this. We use the Shapiro-Wilk test [Shaphiro and Wilk, 1965] on the weights of the 7B LLaMA model [Touvron et al., 2023]. We find that the weights of each hidden unit have different normal distributions. As such, we test the weights of each individual hidden unit. This means for weight $\mathbf{W} \in \mathcal{R}^{in \times out}$ we perform tests over the out dimension. Using a 5% significance threshold, we find that 7.5% of neurons are non-normally distributed which is about 2.5% more than the expected false-positive rate. As such, while almost all pretrained weights appear to be normally distributed there seem to be exceptions. Such exceptions might be due to outliers weights [Dettmers and Zettlemoyer, 2023] or because the p-value of the Shaprio-Wilk test is not accurate for large sample sizes [Shaphiro and Wilk, 1965] that occur in the LLaMA FFN layer hidden units.

8.7 Memory Footprint

Memory Requirement of Parameter-Efficient Finetuning One important point of discussion is the memory requirement of LoRA during training both in terms of the number and size of adapters used. Since the memory footprint of LoRA is so minimal, we can use more adapters to improve performance without significantly increasing the total memory used. While LoRA was designed as a Parameter Efficient Finetuning (PEFT) method, most of the memory footprint for LLM finetuning comes from activation gradients and not from the learned LoRA parameters. For a 7B LLaMA model trained on FLAN v2 with a batch size of 1, with LoRA weights equivalent to commonly used 0.2% of the original model weights[Hu et al., 2021; Liu et al., 2022a], the LoRA input gradients have a memory footprint of 567 MB while the LoRA parameters take up only 26 MB. With gradient checkpointing [Chen et al., 2016], the input gradients reduce to an average of 18 MB per sequence making them more memory intensive than all LoRA weights combined. In comparison, the 4-bit base model consumes 5,048 MB of memory. This highlights that gradient checkpointing is important but also that aggressively reducing the amount of LoRA parameter yields only minor memory benefits. This means we can use more adapters without significantly increasing the overall training memory footprint (see Appendix 8.7 for a detailed breakdown). As discussed later, this is crucial

for recovering full 16-bit precision performance.

The memory footpring for QLoRA training with different LLaMA base models can be seen in Figure 8.6. We see that the 33B model does not quite fit into a 24 GB and that paged optimizers are needed to train it. Depicted is also batch size 1 with a sequence length of 512 and gradient checkpointning. This means, if one uses a larger batch size, or if a long sequence is processed, the activation gradient might consume a considerable amount of memory.

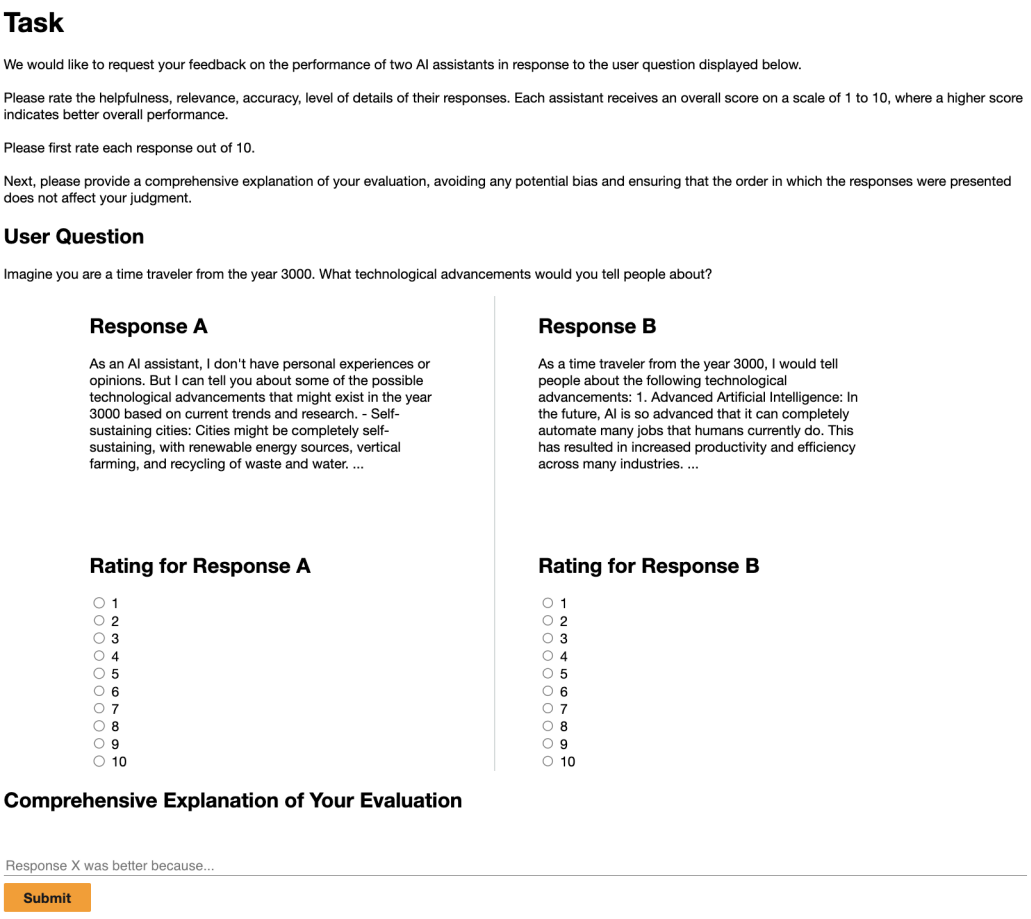

Figure 8.4: The crowdsourcing form used by human annotators.

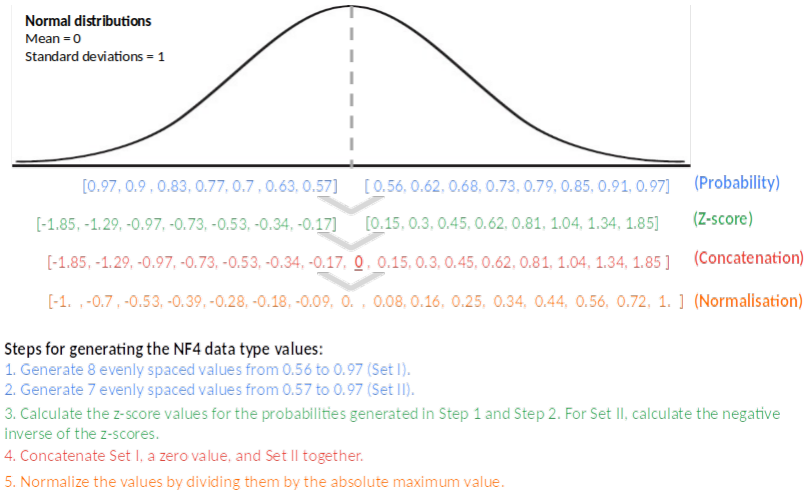

Figure 8.5: Steps required to construct the NF4 data type. Equidistant values of the probability density functions are converted to Z-scores through the quantile function. To ensure the usage of all 16 values the distribution is asymmetric. As such, one half of the normal distribution has 8 and the other 7 probability values. A zero is included to have a discrete zero point. All z-scores and zero are concatenated and then normalized into the range $[-1, 1]$ to receive the final NF4 data type.

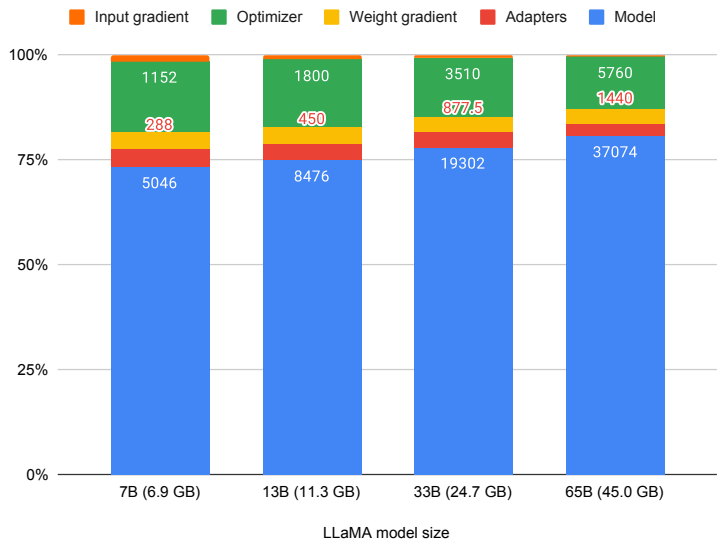

Figure 8.6: Breakdown of the memory footprint of different LLaMA models. The input gradient size is for batch size 1 and sequence length 512 and is estimated only for adapters and the base model weights (no attention). Numbers on the bars are memory footprint in MB of individual elements of the total footprint. While some models do not quite fit on certain GPUs, paged optimizers provide enough memory to allow these models to fit.