So you want your private LLM at home? A survey and benchmark of methods for efficient GPTs

Lukas Tuggener[1,2,3], Pascal Sager[1,5,7], Yassine Taoudi-Benchekroun[6,7], Benjamin F. Grewe[5,6,7], and Thilo Stadelmann[1,4,5]
[1] ZHAW CAI, Winterthur, Switzerland [2] USI, Lugano, Switzerland [3] RwAI AG, Winterthur, Switzerland
[4] ECLT, Venice, Italy [5] AlpineAI AG, Davos, Switzerland [6] ETH AI Center, Zurich, Switzerland
[7] ETH/UZH INI, Zurich, Switzerland
{tugg, sage, stdm}@zhaw.ch, ytaoudi@ethz.ch, bgrewe@ethz.ch

***Abstract*—At least since the introduction of ChatGPT, the abilities of generative large language models (LLMs), sometimes called GPTs, are at the center of the attention of AI researchers, entrepreneurs, and others. However, for many applications, it is not possible to call an existing LLM service via an API due to data protection concerns or when no task-appropriate LLM exists. On the other hand, deploying or training a private LLM is often prohibitively computationally expensive. In this paper, we give an overview of the most important recent methodologies that help reduce the computational footprint of LLMs. We further present extensive benchmarks for seven methods from two of the most important areas of recent progress: model quantization and low-rank adapters, showcasing how it is possible to leverage state-of-the-art LLMs with limited resources. Our benchmarks include resource consumption metrics (e.g. GPU memory usage), a state-of-the-art quantitative performance evaluation as well as a qualitative performance study conducted by eight individual human raters. Our evaluations show that quantization has a profound effect on GPU memory requirements. However, we also show that these quantization methods, contrary to how they are advertised, cause a noticeable loss in text quality. We further show that low-rank adapters allow effective model fine-tuning with moderate compute resources. For methods that require less than 16 GB of GPU memory, we provide easy-to-use Jupyter notebooks that allow anyone to deploy and fine-tune state-of-the-art LLMs on the Google Colab free tier within minutes without any prior experience or infrastructure.**

***Index Terms*—Large Language Models, LlamaV2, fine-tuning, LLM quantization, LLM deployment**

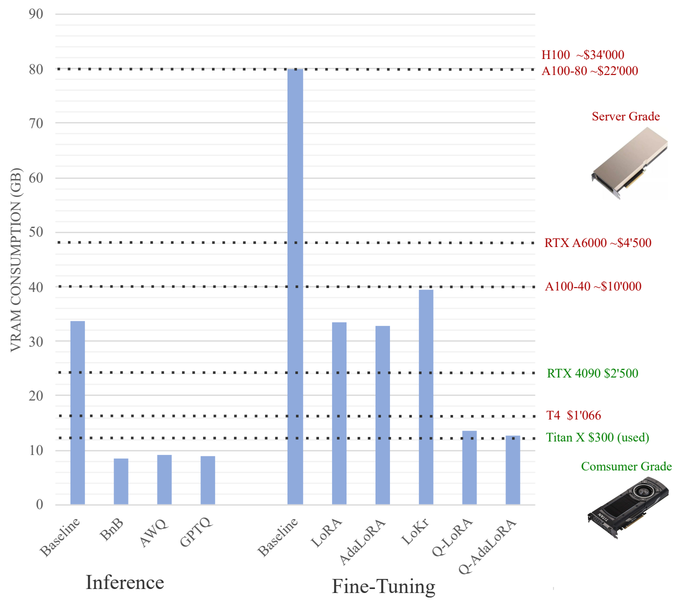

Fig. 1. The VRAM consumption of all benchmarked methods. See Section III-B for inference and Section III-C for fine-tuning. For context, we illustrate the VRAM capacities and list approximate prices of the most popular GPUs in the AI space. Note that GPU prices are very volatile and time dependent.

I. Introduction

Recent Large Language Models (LLMs) have shown remarkable performance on a wide range of language tasks [1], which has earned them an impressive rise in interest [2] that has extended beyond the typical aficionados in academia and industry. Impressive results have been achieved using little prior knowledge and purely leveraging data [3] thanks to the adoption of the transformer architecture [4] (a.k.a. Fast Weight Programmers [5]), which enables each layer of an LLM to focus on specific parts of the input sequence depending on the need. Effectively, models give varying degrees of importance to different elements in the input sequence dynamically, depending on their relevance to the current step in the sequence generation process. This modeling power comes at a cost of crucial computational complexity—transformers scale quadratically with sequence length ($\mathcal{O}(n^2d)$, with n the sequence length, d embedding dimension). However, language transformers only unlock their full potential when having billions of trainable parameters (i.e. putting the L into LLM) [6]. Combined with their computational scaling properties, this makes LLMs notoriously difficult and costly to both train and deploy. Training models for custom applications from scratch requires such large amounts of data and large model sizes that it is only possible for large corporations and universities with substantial resources. Nonetheless, there has been a trend among some of the tech giants to open source even their most capable models, most notably LlamaV2 [7] by Meta. The open availability of these models has enabled a whole new research field concerned with methods that allow the efficient use of pre-trained LLMs on modest hardware configurations.

In this paper, we give an overview of these new technologies by surveying the state of the art. For the two most important among them, quantization (see Section II-A1) and low-rank approximators (see Section II-A2), we present novel benchmark results (see Fig. 1). We evaluate three quantized inference methods and four low-rank approximators for fine-tuning. We benchmark these methods twofold: We first investigate the computational properties of each method, reporting resource consumption and computation speed. In the second step, we investigate the quality of the model outputs by using automated evaluation metrics and conducting a qualitative study on select outputs. The GPU memory (VRAM) requirements of some of the tested methods are low enough that they can be deployed on the free tier of a Google Colab instance. For these methods, we provide easy-to-understand Jupyter notebooks that enable anyone to deploy a state-of-the-art LLM within minutes without any prior knowledge or infrastructure.

The remainder of this paper is structured as follows: in Section II, we give an overview of the key technologies that enable LLMs on limited hardware; in Section III-A, we introduce our benchmark methodology; Sections III-B and III-C contain the benchmark results for model inference and fine-tuning, respectively. In Section III-D, we share our Colab notebooks before we draw conclusions in Section IV.

II. Survey of key technologies

In this section, we present an overview of the innovations that have enabled model fine-tuning, inference, and performance evaluation of LLMs on a moderate compute budget. We omit to survey how LLMs should be trained from scratch as this undertaking remains extremely costly, being out of scope for most organizations and practical applications. For general design patterns of efficient deep learning, see [8], [9].

A. Efficient LLM Execution and Adaptation

1) Quantization: The biggest hurdle to the wide adoption of LLMs is their VRAM requirements. The parameters and activation states of these models are usually stored in a 32- or sometimes 16-bit format. It has been shown that quantization methods that can properly represent the characteristic emergent structures in transformer [4] weights can reduce the parameter and activation formats to an 8-bit representation with minimal loss of arithmetic accuracy [10]. Recently, a 4-bit NormalFloat data type has been introduced that is theoretically optimal for storing normally distributed weights [11].

When quantizing a fully pre-trained transformer model, the quantization can take the explicit structure of the model weights into account, making quantization even more efficient: GPTQ [12] leverages approximate second-order information to generate highly accurate approximations with 3 to 4 bits per weight. Activation-aware Weight Quantization (AWQ) [13] achieves a decrease in quantization error by deriving the importance of each weight through observing their activations before quantization, then protecting the 1% most important weights from being quantized.

When also training a model (not only running inference), the required VRAM is drastically increased due to the variables required by the optimizer. A straightforward way to address this is to also quantize the optimizer's weights [14]. Quantization is very widely adopted and considered to be the most effective method for reducing VRAM consumption.

2) Low-Rank Approximators: While quantization methods are often sufficient to run LLM inference on constrained hardware, VRAM constraints prevent fine-tuning LLMs on a single GPU even when using optimizer quantization. Low-rank adaptation of large language models (LoRA) [15] offers an alternative: LoRA freezes the pre-trained model weights and injects trainable low-rank decomposition matrices into each layer of the Transformer. This way the base network can be run in "inference mode" and only the low-rank matrices are trained as a differential update to the base network. AdaLoRa [16] develops this concept further and assigns parameter budgets to different weight matrices according to an importance metric. LoKr [17] recognizes limited representation power of above low-rank representations and suggests using the Kronecker product instead. Low-rank approximators are the key innovation that democratized LLM fine-tuning. In resource-constrained settings they are mandatory.

3) Pruning: A popular strategy to alleviate the high inference cost of deep neural networks is "model pruning" [18]–[20]: Starting from a pre-trained network, one can iteratively remove redundant parameters and retrain the model while maintaining a test accuracy comparable to the original full-sized network. While Large Language Models are natural candidates for network pruning techniques, they have received little interest from the community compared to the other popular compression techniques such as quantization (see II-A1). A likely explanation for this is that standard pruning methodologies require retraining the network, which is too costly in the case of LLMs. Recent works have however shown promising results with pruning without relying on retraining networks [21], [22].

4) CPU Parameter Offloading: CPU offloading in training deep neural networks involves transferring some computations from the GPU to the CPU to alleviate memory constraints and improve training efficiency. Optimal strategies for offloading activations, gradients, and model parameters to the CPU ultimately enable the training of larger models. Microsoft has been at the forefront of this line of research and introduced the open-source library DeepSpeed [23] as well as ZeRO [24]. This includes a "ZeRO-Offload" feature, which performs optimized CPU parameter offloading, enabling large models with up to 13 billion parameters to be efficiently trained on a single GPU. Smart load sharing between CPUs and GPUs is very important in large multi-GPU or even multi-cluster operations. When working with a single GPU it is usually preferable to compute everything on that device to avoid costly data transfers.

5) Stateless Optimizers: Numerous prevalent optimization algorithms come with the cost of additional parameters. For instance, Adam [25] and AdamW [26] involve the computation of both the moving average of first-order moments (mean of gradients) and second-order moments (uncentered variance of gradients) for each parameter. Consequently, they incur the cost of tripling the parameter count. An alternative to quantizing the optimizer weights (see Sec. II-A1) is to use a stateless optimizer like stochastic gradient descent. The drawback of plain SGD is its typically inferior convergence properties. However, this limitation can be mitigated by introducing a learning rate scheduler [27] that systematically reduces the learning rate, ensuring the attainment of a local minimum. Running a stateless optimizer is an efficient way of reducing VRAM requirements. The incurred loss in convergence speed depends on the problem at hand and it is best to investigate in a preliminary study if it is a worthwhile tradeoff.

6) Gradient Checkpointing: Conventionally, intermediate activations from the forward pass are stored in the VRAM for gradient computation during backpropagation, but this becomes impractical in scenarios with limited memory or very large models. Gradient (or: activation) checkpointing [28], [29] addresses this challenge by selectively caching a subset of crucial intermediate activations for gradient computation, dynamically recomputing the remaining activations during the backward pass. This turns out to make a better trade-off than re-computing all activations: The flexibility of gradient checkpointing in choosing activations for checkpointing allows tailored optimization based on a model's specific memory and computational demands. Gradient checkpointing can help to further reduce memory consumption when reducing the batch size is no longer possible. It comes, however, at a steep cost in training speed.

7) Gradient Accumulation: Gradient accumulation [30] is a technique that allows training on large "virtual batches" that are bigger than the maximum batch size fitting into the available VRAM. This is achieved by accumulating the gradients over multiple batches and only running an optimizer step once enough gradients for the desired "virtual batch size" have been collected. This is often necessary when working with a limited VRAM budget. Gradient accumulation ensures that the necessary batch size can be reached which reduces the noise in the gradient updates enough for the LLMs to properly converge.

8) Software Engineering: LLMs typically operate in an autoregressive fashion [4], predicting the succeeding token based on the prior sequence. Consequently, output generation time increases as the predicted sequence's length grows. In the context of token streaming (as, e.g., in ChatGPT's user interface), the output can be exhibited token by token, as the already produced sequence is directly presented to the user. Furthermore, the autoregressive approach also facilitates continuous batching [31]. In this method, incoming requests are accumulated in a queue, and after each autoregressive step, they are incorporated into the currently processed batch, provided the memory limit permits. Furthermore, sequences that reach the end-of-sequence token are removed from the batch to free up space, allowing other sequences in the queue to be processed.

These measures are crucial when running LLMs in production; Token streaming should be used when the users are actively waiting for the response, allowing users to promptly read the result, even if it hasn't been fully generated. Continuous batching, on the other hand, is particularly beneficial in scenarios where the timing and quantity of requests are uncertain, for example, in multi-user settings.

B. Evaluation

Various metrics such as perplexity [32], BLEU score [33], and F1 score have been proposed to gauge the performance of language models on specific tasks, including translation [34], and text classification [35]. However, LLM-based assistants have started to exhibit more general capabilities across diverse tasks such as writing, chatting, and coding [7], [36]–[38], making evaluating their broad capabilities more challenging. Existing benchmarks to evaluate LLMs include core-knowledge benchmarks [39]–[45] that typically require LLMs to generate a short, specific answer to benchmark questions that can be automatically validated. Instruction-following benchmarks [46]–[50] use more open-ended questions and more diverse tasks to evaluate LLMs after instruction fine-tuning. Conversational benchmarks [51]–[53] focus on dialogue; however, the diversity and complexity of their questions often fall short in challenging the capabilities of the latest models.

Many of these performance metrics primarily center on assessing multi-turn dialogues involving open-ended inquiries, or they fail to pose a sufficient challenge for contemporary cutting-edge models. Although human preferences directly gauge a chatbot's efficacy in multi-turn interactions between humans and AI, this approach lacks scalability. Based on this insight, Zheng et al. [54] propose to bridge this gap by using the novel MT-bench benchmark that tests the core capabilities of state-of-the-art models while leveraging an auxiliary LLM model as a judge. Remarkably, their findings reveal that auxiliary LLM models exhibit a level of agreement comparable to that of human annotators, rendering them suitable for evaluating other LLMs.

In the next section, we apply MT-bench in conjunction with an auxiliary (LLM) to assess the efficacy of a model across diverse tasks. Moreover, this metric proves apt for discerning whether model performance drops during fine-tuning when using, for example, quantization methods, as it not only provides a score per category but also allows pairwise comparison between models.

III. BENCHMARKING PERFORMANCE AND QUALITY

We present benchmarks for two distinct use cases: First, we assess model inference by employing a pre-trained model. Second, we examine fine-tuning on custom data. Model quantization and LoRA are the most promising, impactful, and universally applicable among the plethora of techniques introduced above. Therefore, we chose them for benchmarking. In both scenarios, we conduct a comparative analysis between a given model and the same model incorporating additional quantization techniques. In the inference scenario, we benchmark quantization techniques, while in the fine-tuning scenario, we test multiple low-rank approximations. One focus lies on evaluating VRAM usage, speed, and model performance, as these metrics are pivotal for the effective deployment of models in resource-constrained environments. The second focus lies on gauging the quality of the generated conversations. Since it is the most capable openly available LLM, we use LlamaV2 [7] as the basis for all of our experiments. We use its 7-billion parameter variant – larger versions require a substantial GPU infrastructure, even when paired with the techniques introduced above.

A. Benchmarking methodology

1) Resource usage and speed metrics: For inference, we systematically report metrics acquired during the quantitative evaluation (c.f. Section III-A2) on the MT benchmark. This includes VRAM usage (the most constraining factor for the seamless execution of state-of-the-art models with billions of parameters), generated tokens per second, and overall power consumption in watt-hours (Wh). For fine-tuning, we provide metrics obtained during the re-training, including VRAM usage, processed batches per second, and overall power consumption. For simplicity, we constrain the models to a batch size of 1, even though the aforementioned methods might reduce VRAM requirements to allow an increase in batch size, contingent on available computational resources.

The experiments are conducted using a single Nvidia A-100 40GB GPU. The selection of this GPU was based on its ample VRAM, allowing the execution of all model versions (except direct fine-tuning which is out of the scope of this work), both with and without quantization, facilitating a fair comparison. We also report results generated using a Nvidia T4 16GB GPU to demonstrate the feasibility of running and fine-tuning models on constrained hardware ("at home"). It is important to note that, due to the memory limitations of the Nvidia T4 16GB GPU, only models employing quantization methods could be executed on it.

2) Quantitative Evaluation: We employ the MT-benchmark that contains 80 high-quality multi-turn questions described in [54]. They are manually designed to evaluate the models' proficiency in multi-turn conversation and instruction-following capabilities. The questions are organized into eight categories, each holding ten challenging questions: writing, roleplay, extraction, reasoning, math, coding, STEM knowledge, and humanities/social science knowledge. Each multi-turn question comprises an initial plus a subsequent inquiry.

The multi-turn questions are fed into all model versions, and the resulting responses are stored. Subsequently, OpenAI's GPT-4 model is utilized to assess response quality by assigning a score ranging from 1 to 10 to each answer. To obtain the score, we follow the prompting scheme of Zheng et al. [54], asking GPT-4 to evaluate the model's response for various factors such as "relevance", "accuracy", and "depth" before providing an overall score. The average score per category is then computed to provide a comprehensive evaluation of the model's performance across different categories. In addition to this single-answer evaluation, a pairwise comparison is conducted between the baseline model and a second model version that has been tuned for efficiency. In this process, the responses from both models are presented to the auxiliary LLM, GPT-4, which is then tasked with determining the superior model or declaring a tie. Both procedures adhere to the methodology proposed by Zheng et al. [54].

3) Qualitative Evaluation: As automatic evaluation of generative language models is still in its infancy, it is important to also qualitatively inspect outputs to assess the quality of a model. We systematically do this by prompting our models with the same catalog of five questions sourced from MT-bench. Then, eight individual human judges rate all the outputs on a scale from 1 (useless) to 5 (perfect answer). Finally, we average across the marks given by each judge to obtain an objective score. We use the following five multi-turn questions to assess the ability of the models on a broad range of topics:

Q1 (Writing) Round 1: *Compose an engaging travel blog post about a recent trip to Hawaii, highlighting cultural experiences and must-see attractions.*
Round 2: *Rewrite your previous response. Start every sentence with the letter A.*

Q2 (Reasoning) Round 1: *Thomas is very healthy, but he has to go to the hospital every day. What could be the reasons?*
Round 2: *Can you explain why the above question is interesting?*

Q3 (Math) Round 1: *Benjamin went to a bookstore and purchased a variety of books. He bought* 5 *copies of a sci-fi novel, each priced at* \$20, 3 *copies of a history book priced at* \$30 *each, and* 2 *copies of a philosophy book for* \$45 *each. What was the total cost of his purchases?*
Round 2: *Suppose Benjamin decides to sell each of these books at a 25% markup from the price he purchased them. What would be his total revenue if he sold all the books he bought?*

Q4 (Stem) Round 1: *Photosynthesis is a vital process for life on Earth. Could you outline the two main stages of photosynthesis, including where they take place within the chloroplast, and the primary inputs and outputs for each stage?*
Round 2: *How much energy can a tree produce through photosynthesis in its lifetime? Please provide an estimate using actual numerical values and thoroughly explain your thought process step-by-step.*

Q5 (Humanities) Round 1: *Which methods did Socrates employ to challenge the prevailing thoughts of his time?*
Round 2: *Let's bring Socrates to the modern world. Generate a conversation between Socrates and Bill Gates to debate on generative AI for education.*

B. Results for Efficient Inference

In the subsequent analysis, we assess different quantization techniques with a focus on computational, quantitative, and qualitative performance. Out of the numerous open-source LLMs [1], we selected"LlamaV2-7B" and its dialogue-fine-tuned version "LlamaV2-7B-Chat" as the basis for this evaluation. This choice is based on the models' good performance with a manageable, popular parameter count of "just" 7B [7].

Resource usage and speed metrics. Table I presents an overview of the computational requirements of the LlamaV2-7B-Chat model and its variants employing quantization techniques during the processing of the MT-bench dataset. The first column shows the VRAM usage. It is important to acknowledge that VRAM utilization may vary based on the sampled model response that is reused as conversational context in subsequent follow-up questions. The adoption of

TABLE I
VRAM USAGE, TOKENS PROCESSED PER SECOND, AND THE OVERALL POWER CONSUMPTION DURING INFERENCE ON MT-BENCH. THE MODEL USED IS LLAMAV2-CHAT, OPTIONALLY COMBINED WITH BITS-AND-BYTES (BNB), NESTED BNB, AWQ, AND GPTQ QUANTIZATION TECHNIQUES.

	VRAM usage [GB]	avg. tokens/s		power consumption [Wh]	
GPU	A100	A100	T4	A100	T4
LlamaV2-7B-Chat	33.80	36.06	out of memory	92.39	out of memory
LlamaV2-7B-Chat BnB	8.40	25.45	16.07	63.53	57.09
LlamaV2-7B-Chat AWQ	9.10	86.66	26.32	33.72	38.96
LlamaV2-7B-Chat GPTQ	9.02	34.20	28.33	52.56	32.28

quantization techniques enables the execution of the model on GPUs with constrained VRAM, such as the Nvidia T4, as the VRAM requirements reduce from 33.80GB to below 10GB. Alternatively, in scenarios where sufficient memory is available, this facilitates the utilization of batch sizes that are three to four times larger, thereby enabling the concurrent processing of a higher volume of requests.

When examining the tokens processed per second using the A-100 GPU, it becomes evident that quantization via BnB (25.45 tokens/s) exhibits significantly slower performance compared to the baseline without quantization (36.06 tokens/s). Thus, squeezing parameters and activations into lower precision datatypes reduces memory requirements on the cost of speed. In contrast, more advanced algorithms such as GPTQ and AWQ can even preserve or improve the throughput while having a reduced memory footprint: GPTQ quantization achieves a throughput of 34.2 tokens/s, nearly matching the baseline. Remarkably, AWQ quantization attains an impressive throughput of 86.66 tokens/s, approximately 2.4 times faster than the baseline. Unfortunately, this enhanced performance does not scale to the T4 GPU due to this GPU's reliance on the older Turing architecture. The T4 GPU lacks the novel CUDA kernels of the Ampere architecture, which is extensively utilized by AWQ. Consequently, AWQ on the T4 GPU achieves performance similar to GPTQ. However, as previously elucidated, the quantization methods enable larger batch sizes and the concurrent processing of more tokens, thereby enhancing overall throughput. Thus, even slower methods like BnB could outperform the baseline when quantization is employed to increase the batch size.

Quantization techniques also allow for substantially diminishing power consumption. This is attributed to the utilization of lower-precision datatypes for representing model parameters and activations, resulting in enhanced computational efficiency. On the A-100 GPU, the standard model consumes 92.39Wh, whereas BnB has a power consumption of 63.53Wh, and nested BnB consumes 69.15Wh. GPTQ and AWQ employ similar datatypes as BnB, but due to their higher throughput, these methods achieve an even lower total power consumption of 33.72Wh (AWQ) and 52.56Wh (GPTQ), respectively. Unfortunately, due to the aforementioned reasons, AWQ struggles to transfer its high efficiency to the older GPU.

To summarise, from a purely computational perspective, AWQ quantization is preferable if the GPU is equipped with compatible CUDA drivers; otherwise, GPTQ is recommended. These approaches demand significantly less VRAM compared to the baseline model while maintaining or increasing processing speed, even when the batch size remains unchanged. Unfortunately, AWQ and GPTQ cannot be used for model training, making BnB the preferred method for quantization while learning model weights.

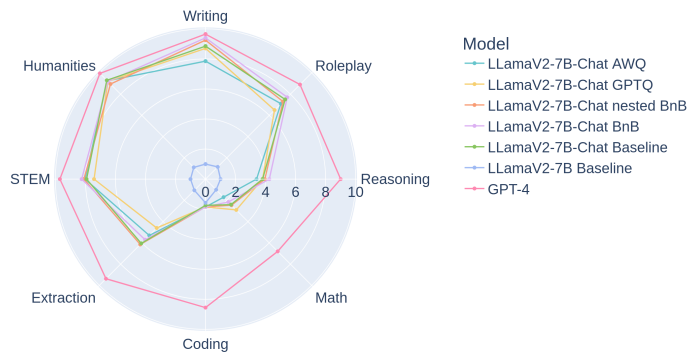

Fig. 2. Single score evaluation results for LlamaV2-7B-Chat with different quantization techniques and the state-of-the-art GPT-4 for comparison.

Quantitative evaluation. To quantitatively assess the influence of quantization on LlamaV2-7B-Chat's output quality, we utilize the single-score judgment based on MT-bench. Additionally, we include evaluations of LlamaV2-7B as a lower bound and GPT-4 as an upper bound. Please note that GPT-4, when used as a judge, favors its own answers with a 10% bias in pair-comparisons, suggesting that it rates itself also slightly better than other models in single answer grading [54]. LlamaV2-7B is used as the lower bound due to its identical architecture but lack of fine-tuning for dialogue, a requirement for MT-bench. The findings are presented in Figure 2. As expected, the LlamaV2-7B-Chat model and its quantized versions are noticeably better than the lower bound and worse than the upper bound. There is only a marginal performance difference between the baseline and the quantized models, suggesting that the application of quantization techniques neither degrades nor enhances performance.

Qualitative evaluation. Table II shows the average grade for the answers of all the quantized versions of LlamaV2-Chat to our five selected questions (see Section III-A3). Notably, the non-quantized model achieves the highest average grade of 3.35, surpassing GPTQ with an average grade of 3.1 and BnB with an average grade of 2.9. In contrast, AWQ received a lower average rating of 1.95 compared to the other quantization methods. This discrepancy suggests that humans

TABLE II
AVERAGE (ACROSS 8 HUMAN RATERS) USEFULNESS EVALUATION (1=USELESS, 5=PERFECT) OF ANSWERS FROM QUANTIZED VERSIONS OF LLAMAV2-CHAT

	Q1	Q2	Q3	Q4	Q5	Average
no quantization	3.75	1.25	4.5	4	3.25	3.35
BnB	2.25	2	2.75	3.5	4	2.9
AWQ	1.5	1.5	1.5	2.5	2.75	1.95
GPTQ	1.25	1.75	4.75	3.75	4	3.1

do not favor AWQ as a quantization technique, which is in contrast to the outcomes of the quantitative analysis, where no performance gap could be observed. Our results indicate that quantization techniques improve computational efficiency but degrade the overall dialogue quality.

C. Results for Efficient Model Fine-Tuning

When there is no task-appropriate LLM available, it becomes necessary to fine-tune an LLM on a new dataset. To evaluate the effect of efficiency techniques for LLM fine-tuning, we fine-tune a LlamaV2 base model on the dataset that was designed for the training of Guancano [11] and that is a subset of the OpenAssistant [53] dataset. Thus, we train the model to behave like a chat-based assistant rather than a general generative model. While this is of no practical importance as a new model due to the existence of LlamaV2-Chat, it allows us to keep a consistent evaluation scheme across all experiments for this benchmarking. Fine-tuning a LlamaV2 model without a low-rank adapter is only possible with very large computational resources (at least 80GB VRAM on a single chip), therefore we consider the widely used LoRA [15] method as our baseline. Additionally, we fine-tune models using AdaLoRA [16] and LoKr [17]. We also test quantized versions of LoRA (Q-LoRA) and AdaLoRA (Q-AdaLoRA), which provide a further reduction in VRAM requirement. We train our models with a constant learning rate of 0.0001 and a weight decay of 0. We use a batch size of 1 and 8 gradient accumulation steps. We cap the gradient norm at 0.3 and train for $15,000$ steps. LoKr has a relatively heavy VRAM load, therefore we have to reduce the maximum sequence length to 400 tokens instead of the default of 500, even on a 40GB GPU. To enable fair comparisons, we keep this training configuration (including max sequence length) consistent across all methods. The full training details can be found in the public WandB logs linked here: https://github.com/tuggeluk/LMM_at_home. We have opted for this relatively short fine-tuning due to our limited computational resources. This setup allows us to compare the different low-rank adaptation methods, but the resulting models are not expected to be competitive with the highly tuned LlamaV2-Chat.

Resource usage and speed metrics. Table III shows an overview of the resources consumed by the different fine-tuning methods for the A100 and T4 GPUs. Among the non-quantized adapters, LoKr requires a higher amount of memory than LoRA and AdaLoRA. Quantizing the adapter slashes the memory required for fine-tuning by almost two-thirds, allowing the use of T4 chips that only have 16GB of memory. The processing speed does not deviate too sharply between the methods with the fastest processing (LoRA: 0.217 steps/s) and the slowest processing (LoKr: 0.15 steps/s) on an A-100. Quantizing causes a slowdown of roughly 20% in both cases, comparable to the BnB quantization during inference. The T4 is much slower at 0.04 steps/s for both methods, yielding an increase in training time of more than four-fold. The overall power consumption on the A-100 is highly correlated with the training time, while the reduced memory usage of the quantized methods has a slim effect on the energy consumption, making the faster, unquantized training a more energy-efficient option. While the T4 has a much lower peak power consumption, this is largely compensated by longer training time, leading to similar energy usage for both GPU models.

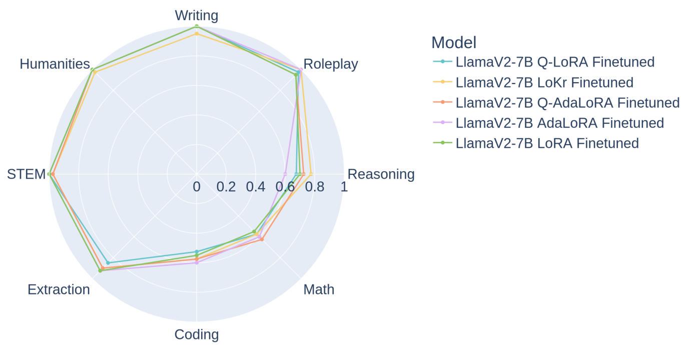

Fig. 3. Pair-wise comparison results of LlamaV2-7B vs. fine-tuned models.

Quantitative evaluation. To assess the impact of efficiency-optimized fine-tuning on chat quality, we conduct a pair-wise comparison using MT-bench. In this comparison, a judging model assigns scores of 1.0 if the fine-tuned version is superior to the baseline model, 0.5 if they perform equally, and 0.0 if the fine-tuned version is inferior. The results, depicted in Figure 3, reveal that our fine-tuned models consistently outperform the baseline, consisting of a stock LlamaV2, across all categories. This underscores the effectiveness of our fine-tuning. Notably, the fine-tuned models exhibit especially superior performance in the categories of writing, roleplay, extraction, STEM, and humanities, with average scores hovering around 1.0, indicating their consistent superiority over the baseline in nearly every instance.

In addition to conducting pairwise comparisons, we present outcomes based on single-score assessments using MT-bench. The outcomes are illustrated in Figure 4. The results from our fine-tuned models follow a similar pattern as the results from the quantized LlamaV2-Chat version used for inference (cf. Figure 2), i.e., work better in specific categories than others. However, overall, they perform worse than the models used during inference. This discrepancy arises from our use of a simplified and short fine-tuning approach, wherein we refrain from manually evaluating multiple checkpoints to identify the optimal model version. Nonetheless, this validation affirms

TABLE III
VRAM USAGE, TOKENS PROCESSED PER SECOND, AND OVERALL POWER CONSUMPTION DURING FINE-TUNING. THE BASE MODEL USED IS LLAMAV2-7B, FINE-TUNED WITH LOW-RANK APPROXIMATORS LORA, ADALORA, LOKR, QUANTIZED LORA, AND QUANTIZED ADALORA.

	VRAM usage [GB]	steps/s		power usage [Wh]	
GPU		A100	T4	A100	T4
LoRA	33.4	0.217	out of memory	6,641	out of memory
AdaLoRA	32.7	0.203	out of memory	6,560	out of memory
LoKr	39.5	0.15	out of memory	10,080	out of memory
Q-LoRA	13.5	0.178	0.04	7,990	7,402
Q-AdaLoRA	12.7	0.168	0.04	7,936	7,843

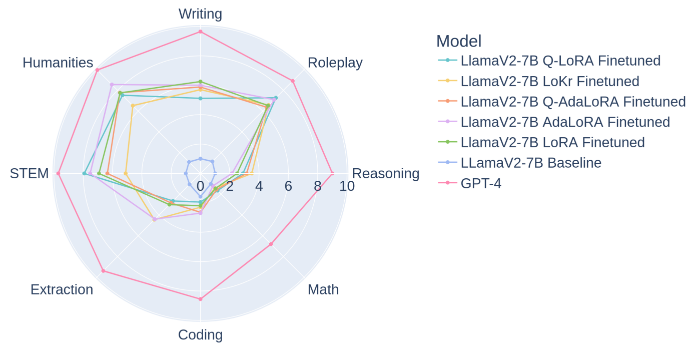

Fig. 4. Single score evaluation results for pre-trained LlamaV2-7B (baseline) and our fine-tuned versions.

that our results are comparable to the open-source fine-tuned versions.

When comparing the results from the fine-tuned versions, only subtle distinctions are evident. For instance, LoKr exhibits a slightly inferior performance in tasks related to humanities and STEM compared to the other models. Conversely, LoKr, along with AdaLora, surpasses other models in tasks involving information extraction. Moreover, quantization methods appear to have negligible effects on the overall quality of the model. The mean rating for LoRA stands at 4.58, and its quantized counterpart, Q-LoRA, is rated at 4.63 $(+0.05)$. Similarly, AdaLoRA achieves an average score of 4.96, and its quantized variant, Q-AdaLoRA, is rated at 4.60 (-0.36). These variances are minimal, especially considering the scale of ratings ranging from one to ten.

Qualitative evaluation. Table IV shows the average grade for answers of all fine-tuned models to our five selected questions (see Section III-A3). We observe that, on average, LoRA performs the best, with a strong performance gap to AdaLoRA that in turn outperforms LoKr. Interestingly, the human evaluation of the answers does indicate a quite steep drop-off in quality between the quantized and unquantized versions of LoRA and AdaLoRA. This is a further indication that model quantization does not incur just a negligible loss in performance as most automated evaluations suggest.

D. Colab Notebooks to “Run Your LLM at Home”

We provide easy-to-use Colab notebooks for all presented methods that work on a T4 GPU (see https://github.com/tuggeluk/LMM_at_home). Using these notebooks, all quantized inference methods as well as Q-LoRA and Q-AdaLoRA fine-tuning can be easily deployed on a Colab free-tier instance by anyone without any coding.

TABLE IV
AVERAGE (ACROSS 8 HUMAN RATERS) USEFULNESS EVALUATION (1=USELESS, 5=PERFECT) OF ANSWERS FROM FINE-TUNED LLAMAV2S.

	Q1	Q2	Q3	Q4	Q5	Average
LoRA	2.5	2.25	0.75	3.75	3.25	2.5
Q-LoRA	2	0.75	2.25	3.25	1.75	2
AdaLoRA	1.5	1.25	2.5	3	2.25	2.1
Q-AdaLoRA	1.25	1.75	1.25	1.5	2.25	1.6
LoKr	2.25	1.5	1	0.5	3	1.65

IV. CONCLUSIONS

We have surveyed the state of the art in methods for efficient fine-tuning and inference with LLMs, and have shown that the current ecosystem of open-source LLMs with accompanying resource-sparse technologies allows for efficient and effective LLM inference and fine-tuning with limited computational resources. As expected, quantization led to a massive reduction in GPU memory requirements. For inference, this has also carried over to a significant reduction in power consumption which might be an important factor for large operations. Most quantization methods claim to cause a negligible loss in performance. However, we have been able to show, to the best of our knowledge for the first time, that quantization has caused a noticeable degradation in performance for both inference and fine-tuning on LlamaV2 models. Important to note here is that the automated evaluation was not able to accurately capture this shift in performance, rather, it was only found thanks to the qualitative evaluation using human input. This further enforces an emerging sentiment in the community that the current evaluation methods for generative LLMs are not sufficient [55]. We have further shown that low-rank-adaptation-based fine-tuning effectively trains LLMs. AdaLoRA and LoKr were not able to outperform the default of LoRA (again in the context of LlamaV2 models), which calls into question whether these further developments are useful, especially the computationally heavy LoKr. We furthermore observed that our fine-tuned models with short training perform worse than LlamaV2-Chat, which shows that fine-tuning still is a lengthy process and highly-tuned open source models are usually preferable over home-brewed variants where available.

Acknowledgments

This work was partially funded by a DIZH Fellowship of LT. We thank Thomas Graf, Pascal Schneider, Simon Spalinger, and Daniel Tuggener for contributing as LLM raters.

References

[1] W. X. Zhao *et al.*, "A Survey of Large Language Models," Nov. 2023. arXiv:2303.18223.

[2] J. Segessenmann, T. Stadelmann, A. Davison, and O. Dürr, "Assessing deep learning: a work program for the humanities in the age of artificial intelligence," *AI and Ethics*, pp. 1–32, 2023.

[3] T. Stadelmann, T. Klamt, and P. H. Merkt, "Data centrism and the core of data science as a scientific discipline," *Archives of Data Science, Series A*, vol. 8, no. 2, 2022.

[4] A. Vaswani *et al.*, "Attention is All you Need," in *Proc. of NIPS'17*, vol. 30.

[5] I. Schlag, K. Irie, and J. Schmidhuber, "Linear Transformers Are Secretly Fast Weight Programmers," in *Proc. of ICML'21*, vol. 139, pp. 9355–9366.

[6] J. Kaplan *et al.*, "Scaling Laws for Neural Language Models," Jan. 2020. arXiv:2001.08361.

[7] H. Touvron *et al.*, "Llama 2: Open Foundation and Fine-Tuned Chat Models," July 2023. arXiv:2307.09288.

[8] G. Menghani, "Efficient deep learning: A survey on making deep learning models smaller, faster, and better," *ACM Computing Surveys*, vol. 55, no. 12, pp. 1–37, 2023.

[9] L. Tuggener, M. Amirian, F. Benites, P. von Däniken, P. Gupta, F.-P. Schilling, and T. Stadelmann, "Design patterns for resource-constrained automated deep-learning methods," *AI*, vol. 1, no. 4, pp. 510–538, 2020.

[10] T. Dettmers, M. Lewis, Y. Belkada, and L. Zettlemoyer, "LLM.int8(): 8-bit Matrix Multiplication for Transformers at Scale," in *NeurIPS'22*, vol. 35, pp. 30318–30332.

[11] T. Dettmers, A. Pagnoni, A. Holtzman, and L. Zettlemoyer, "QLoRA: Efficient Finetuning of Quantized LLMs," vol. 36, pp. 10088–10115.

[12] E. Frantar, S. Ashkboos, T. Hoefler, and D. Alistarh, "GPTQ: Accurate Post-Training Quantization for Generative Pre-trained Transformers," Proc. of ICLR'23.

[13] J. Lin *et al.*, "AWQ: Activation-aware Weight Quantization for LLM Compression and Acceleration," Proc. of MLSys'24.

[14] T. Dettmers, M. Lewis, S. Shleifer, and L. Zettlemoyer, "8-bit Optimizers via Block-wise Quantization," in *Proc. of ICLR'22*.

[15] E. J. Hu *et al.*, "LoRA: Low-Rank Adaptation of Large Language Models," in *Proc. of ICLR'22*.

[16] Q. Zhang *et al.*, "AdaLoRA: Adaptive Budget Allocation for Parameter-Efficient Fine-Tuning," Proc. of ICLR'23.

[17] A. Edalati *et al.*, "KronA: Parameter Efficient Tuning with Kronecker Adapter," Dec. 2022. arXiv:2212.10650.

[18] Y. LeCun, J. Denker, and S. Solla, "Optimal Brain Damage," *Proc. of NIPS'89*, vol. 2, 1989.

[19] B. Hassibi, D. G. Stork, and G. J. Wolff, "Optimal brain surgeon and general network pruning," in *Proc. of the IEEE ICNN'93*, pp. 293–299.

[20] S. Han, H. Mao, and W. J. Dally, "Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding," Proc. of ICLR'15.

[21] M. Sun, Z. Liu, A. Bair, and J. Z. Kolter, "A Simple and Effective Pruning Approach for Large Language Models," Oct. 2023. arXiv:2306.11695.

[22] E. Frantar and D. Alistarh, "SparseGPT: Massive Language Models Can Be Accurately Pruned in One-Shot," in *Proc. of ICML'23*, pp. 10323–10337.

[23] J. Rasley, S. Rajbhandari, O. Ruwase, and Y. He, "DeepSpeed: System Optimizations Enable Training Deep Learning Models with Over 100 Billion Parameters," in *Proc. of the ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pp. 3505–3506, 2020.

[24] S. Rajbhandari, J. Rasley, O. Ruwase, and Y. He, "ZeRO: Memory Optimizations Toward Training Trillion Parameter Models," in *Proc. of SC'20*, pp. 1–16.

[25] D. P. Kingma and J. Ba, "Adam: A Method for Stochastic Optimization," in *Proc. of ICLR'15*.

[26] I. Loshchilov and F. Hutter, "Decoupled Weight Decay Regularization," in *Proc. of ICLR'19*.

[27] I. Loshchilov and F. Hutter, "SGDR: Stochastic Gradient Descent with Warm Restarts," in *Proc. of ICLR'17*.

[28] T. Chen, B. Xu, C. Zhang, and C. Guestrin, "Training Deep Nets with Sublinear Memory Cost," Apr. 2016. arXiv:1604.06174.

[29] A. Gruslys *et al.*, "Memory-Efficient Backpropagation through Time," in *Proc. of NIPS'16*, pp. 4132–4140.

[30] S. Gugger, L. Debut, T. Wolf, P. Schmid, Z. Mueller, S. Mangrulkar, M. Sun, and B. Bossan, "Accelerate documentaion," 2022.

[31] G.-I. Yu *et al.*, "Orca: A Distributed Serving System for Transformer-Based Generative Models," in *USENIX Symposium on Operating Systems Design and Implementation*, pp. 521–538, July 2022.

[32] F. Jelinek, R. L. Mercer, L. R. Bahl, and J. K. Baker, "Perplexity–a measure of the difficulty of speech recognition tasks," *The Journal of the Acoustical Society of America*, vol. 62, pp. S63–S63, Dec. 1977.

[33] K. Papineni, S. Roukos, T. Ward, and W.-J. Zhu, "Bleu: a Method for Automatic Evaluation of Machine Translation," in *Proc. of ACL'02*, pp. 311–318.

[34] S. P. Singh *et al.*, "Machine translation using deep learning: An overview," in *International Conference on Computer, Communications and Electronics*, Comptelix, pp. 162–167, July 2017.

[35] S. Minaee *et al.*, "Deep Learning–based Text Classification: A Comprehensive Review," *ACM Computing Surveys*, vol. 54, pp. 1–40, Apr. 2022.

[36] R. Anil *et al.*, "PaLM 2 Technical Report," Sept. 2023. arXiv:2305.10403.

[37] S. Bubeck *et al.*, "Sparks of Artificial General Intelligence: Early experiments with GPT-4," Apr. 2023. arXiv:2303.12712.

[38] A. Srivastava *et al.*, "Beyond the Imitation Game: Quantifying and extrapolating the capabilities of language models," June 2023. arXiv:2206.04615.

[39] P. Clark *et al.*, "Think you have Solved Question Answering? Try ARC, the AI2 Reasoning Challenge," Mar. 2018. arXiv:1803.05457.

[40] R. Zellers *et al.*, "HellaSwag: Can a Machine Really Finish Your Sentence?," in *Proc. of ACL'19*, pp. 4791–4800, 2019.

[41] D. Hendrycks *et al.*, "Measuring Massive Multitask Language Understanding," Proc. of ICLR'21.

[42] K. Sakaguchi, R. L. Bras, C. Bhagavatula, and Y. Choi, "WinoGrande: An Adversarial Winograd Schema Challenge at Scale," *Communications of the ACM*, vol. 64, pp. 99–106, Aug. 2021.

[43] M. Chen *et al.*, "Evaluating Large Language Models Trained on Code," July 2021. arXiv:2107.03374.

[44] K. Cobbe *et al.*, "Training Verifiers to Solve Math Word Problems," Nov. 2021. arXiv:2110.14168.

[45] W. Zhong *et al.*, "AGIEval: A Human-Centric Benchmark for Evaluating Foundation Models," Sept. 2023. arXiv:2304.06364.

[46] J. Wei *et al.*, "Finetuned Language Models Are Zero-Shot Learners," in *Proc. of ICLR'22*.

[47] S. Mishra, D. Khashabi, C. Baral, and H. Hajishirzi, "Cross-Task Generalization via Natural Language Crowdsourcing Instructions," in *Proc. of ACL'22*, pp. 3470–3487.

[48] Y. Wang *et al.*, "Super-NaturalInstructions: Generalization via Declarative Instructions on 1600+ NLP Tasks," in *Proc. of EMNLP'22*, pp. 5085–5109, Dec.

[49] S. Longpre *et al.*, "The Flan Collection: Designing Data and Methods for Effective Instruction Tuning," vol. 202 of *Proc. of ICML'23*, pp. 22631–22648.

[50] Y. Wang *et al.*, "Self-Instruct: Aligning Language Models with Self-Generated Instructions," in *Proc. of ACL'23*, pp. 13484–13508.

[51] S. Reddy, D. Chen, and C. D. Manning, "CoQA: A Conversational Question Answering Challenge," *Trans. of ACL*, vol. 7, pp. 249–266, 2019.

[52] J. Feng *et al.*, "MMDialog: A Large-scale Multi-turn Dialogue Dataset Towards Multi-modal Open-domain Conversation," Dec. 2022. arXiv:2211.05719.

[53] A. Köpf *et al.*, "OpenAssistant Conversations – Democratizing Large Language Model Alignment," Oct. 2023. arXiv:2304.07327.

[54] L. Zheng *et al.*, "Judging LLM-as-a-Judge with MT-Bench and Chatbot Arena," Oct. 2023. arXiv:2306.05685.

[55] J. Deriu, D. Tuggener, P. Von Däniken, and M. Cieliebak, "Probing the Robustness of Trained Metrics for Conversational Dialogue Systems," in *Proc. of ACL'22*, pp. 750–761.