L1RA: Dynamic Rank Assignment in LoRA Fine-Tuning

Raul Singh[*], **Nicolò Brunello**[*], **Vincenzo Scotti**[†] and **Mark James Carman**[*]
[*]DEIB, Politecnico di Milano
Via Ponzio 34/5, 20133, Milano (MI), Italy
[†]KASTEL, Karlsruhe Institute of Technology (KIT)
Am Fasanengarten 5, 76131, Karlsruhe, Germany
raul.singh@mail.polimi.it nicolo.brunello@polimi.it
vincenzo.scotti@kit.edu mark.carman@polimi.it

Abstract

The ability of Large Language Models (LLMs) to solve complex tasks has made them crucial in the development of AI-based applications. However, the high computational requirements to fine-tune these LLMs on downstream tasks pose significant challenges, particularly when resources are limited. In response to this challenge, we introduce L1RA, a novel technique aimed at dynamically distributing the rank of low-rank adapters during fine-tuning using LoRA. Given a rank budget (i.e., total sum of adapters rank), L1RA leverages L_1 regularisation to prune redundant ranks and redistribute them across adapters, thereby optimising resource utilisation. Through a series of comprehensive experiments, we empirically demonstrate that L1RA maintains comparable or even reduced computational overhead compared to other LoRA variants, including the vanilla approach, while achieving same or better performances. Moreover, the post-training analysis of rank distribution unveiled insights into the specific model components requiring the most adaptation to align with the task objective: the feed-forward layers and the attention output projection. These results highlight the efficacy of L1RA in not only enhancing the efficiency of LLM fine-tuning, but also in providing valuable diagnostic information for model refinement and customisation. In conclusion, L1RA stands as a promising technique for advancing the performance and interpretability of LLM adaptation, particularly in scenarios where computational resources are constrained.

1 Introduction

Large Language Models (LLMs) have revolutionised *Natural Language Processing* (NLP) and *Artificial Intelligence* (AI) (Zhao et al., 2023), enabling sophisticated applications. LLM's language understanding and generation capabilities make them suitable for an impressive number of applications (Raffel et al., 2020; Brown et al., 2020; Sanh et al., 2022). Moreover, their adoption as core for *chatbots* (Scotti et al., 2024) have made them essential for the final consumers of this technology. However, to excel in these specific tasks, even conversation, LLMs often require *fine-tuning*, a process essential for tailoring their vast pre-trained knowledge to new specific contexts and domains. This adaptation ensures optimal performance and task alignment, making fine-tuning a critical step in deploying LLMs effectively.

The fine-tuning process, however, presents challenges, particularly concerning computational resources. Adaptation to specific domains, such as chatbot dialogue or instruction-following tasks, demands substantial computational power, which may be impractical or unfeasible in resource-constrained environments. Recent advancements in efficient fine-tuning techniques, including *Low-Rank Adaptation* (LoRA) (Hu et al., 2022), *prefix tuning* (Li and Liang, 2021) and the *gradient-free methods* like *Memory-efficient Zeroth-order Optimiser* (MeZO) (Malladi et al., 2023), offer promising solutions. These techniques leverage strategies like low-rank parameterisation to reduce computational overhead, making fine-tuning more accessible.

In this paper, we introduce *L_1-regularised Rank Assignment* (L1RA): a technique aimed at enhancing the efficiency and effectiveness of LLM fine-tuning. L1RA extends LoRA by introducing L_1 regularisation to enforce rank sparsity and dynamic rank allocation during training to get the best from the available resources. Assuming a given *rank budget* (i.e., total sum of LoRA adapter ranks), L1RA prunes redundant ranks and reallocates them across adapters during the fine-tuning process. We pair L1RA with our tool *Memory GPU Estimation of LLM Allocation for Training Optimisation* (Memory-GELATO) to be sure to match available resources constraints. Through a series of experiments, ranging from small-scale analyses to comprehensive comparisons

with other fine-tuning techniques, we evaluate the performance of L1RA. The results highlight how L1RA can offer better comparable results to alternative LoRA variants reallocating ranks with negligible difference in resources consumption and better results even with respect to regular LoRA.

We divide the rest of the paper into the following sections. In Section 2, we present the related works on efficient LLM fine-tuning. In Section 3, we explain the reasons behind our work. In Sections 4 and 5, we describe, respectively, the L1RA fine-tuning algorithm and the MEMORY-GELATO tool. In Section 6, we outline the experiments to evaluate our model and in Section 7 we present the obtained results. In Section 8 we comment on the results we obtained. Finally, in Section 9, we sum up our work and suggest possible future extensions.

2 Related works

Efficient fine-tuning techniques have garnered increasing attention lately, due to the computational demands associated with adapting LLMs to specific tasks. The proposed techniques evolved significantly during the last few years. Initial approaches like *Transformer Adapters* (Houlsby et al., 2019; Bapna and Firat, 2019) introduced additional parameters in the form of a pair of linear projections with a bottleneck in the middle, increasing network depth and latency, thereby hindering scalability. In response, LoRA-based solutions (Hu et al., 2022) have emerged as a promising alternative. LoRA addresses the limitations of adapters by introducing low-rank parameterisation, effectively reducing the number of parameters needed for adaptation. This technique has gained widespread adoption for its ability to achieve efficient fine-tuning without compromising performance. Alternative techniques like MeZO (Malladi et al., 2023) target the training algorithm rather than the network structure, focusing on fine-tuning through forward passes only, eliminating the need for backpropagation and the subsequent overhead. Other approaches like prefix-tuning (Li and Liang, 2021) learn only the embeddings of a *continuos prompt* that can be used as a prefix to the input to condition the LLM output towards the desired task. Among these techniques, LoRA stands out as the most adopted due to its effectiveness in balancing computational efficiency, performance and ease of use.

As premised, LoRA operates by introducing pairs of low-rank matrices $\mathbf{A} \in \mathbb{R}^{d_{in} \times r}$ and $\mathbf{B} \in \mathbb{R}^{r \times d_{out}}$ into the network architecture (see Figure 1); the product $\Delta\mathbf{W} \in \mathbb{R}^{d_{in} \times d_{out}}$ of these two matrices encodes the weights difference induced by fine-tuning for a specific weight matrix $\mathbf{W} \in \mathbb{R}^{d_{in} \times d_{out}}$ of the pre-trained model, as explained by Equation (1). During fine-tuning, these adapter matrices are updated while the original LLM parameters are kept frozen. By leveraging low-rank parameterisation, LoRA effectively reduces the computational overhead associated with fine-tuning while preserving the expressive power of the LLM. Moreover, this approach has demonstrated empirically significant improvements in efficiency without sacrificing performance across various downstream tasks.

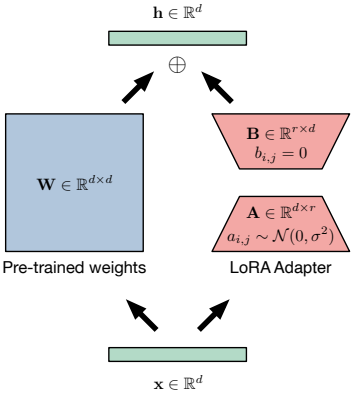


Figure 1: LoRA adapters: pre-trained weights are frozen while the two adapter matrices are updated during the fine-tuning.

$$\mathbf{h} = \mathbf{x} \cdot (\mathbf{W} + \Delta\mathbf{W}) = \mathbf{x} \cdot \mathbf{W} + \mathbf{x} \cdot (\mathbf{A} \cdot \mathbf{B}) \quad (1)$$

While LoRA offers notable benefits, several variants have been proposed to address specific limitations or further enhance its capabilities. Examples of these alternative solutions are those aimed at stabilising the training process, like LoRA+ (Hayou et al., 2024), which introduces a matrix-specific scaling parameter on the learning rate to improve performances and convergence time, and *Rank-Stabilised* LoRA (rsLoRA) (Kalajdzievski, 2023), which uses a rank correcting factor to prevent gradient collapse. Other variants, like *Quantised* LoRA (QLoRA) (Dettmers et al., 2023), aim at further reducing computational complexity by heavily quantising the base model weights (for reduced memory requirements and increased inference speed) while operating on floating-point representation of the trainable weights (for numeric precision and, thus, training stability), thereby improving the overall efficiency. In this paper we focus on techniques for

adaptive rank allocation. In fact, LoRA adapters depend on the rank hyper-parameter, which can be selected dynamically for each pair of adapter matrices. In this sense, some solutions have been proposed to tackle the issue of rank selection in order to (i) get rid of unused parameters and (ii) find the best possible rank allocation allowed by the available memory.

One of the first solutions for dynamic rank allocation was presented with AdaLoRA (Zhang et al., 2023), which enforces a *Singular Value Decomposition*-inspired (SVD-inspired) decomposition of the adapter weights through additional regularisation terms in the loss. Further refinements of this technique came with *Sparse* LoRA (SoRA) (Ding et al., 2023), which uses an intermediate gating mechanism with L_1 regularisation and *proximal gradient descent* to iteratively reduce the used ranks, and, *Vector-based Random matrix Adaptation* (VeRA) (Kopiczko et al., 2023), which reduces the trainable LoRA parameters through shared random weights matrices and works on rank allocation updating only layer-specific parameter vectors. In parallel, *Dynamic rank selection* LoRA (DyLoRA) (Valipour et al., 2023), proposed a solution exploring a range of possible ranks during training to find the optimal ones for each matrix.

3 Motivations

LoRA adapters represent a valuable step towards end-user fine-tuning of LLMs, making this technology more accessible and customisable. The existence of techniques like AdaLoRA, SoRA or DyLoRA allowing for dynamic rank and pruning (i.e., removing the i-th column in $\mathbf{A}$ and the i-th row in $\mathbf{B}$) are the results of advances towards better exploitation of computational resources. Hereafter, we highlight some points of improvement for AdaLoRA and SoRA (the main solutions for dynamic rank allocation), in terms of computational resources exploitation, that are motivating our work.

AdaLoRA proposes a SVD-inspired formulation of the adapter:

$$\Delta\mathbf{W} = \mathbf{U}\cdot\mathbf{\Sigma}\cdot\mathbf{V}^\top = \mathbf{U}\cdot\text{diag}(\boldsymbol{\sigma})\cdot\mathbf{V}^\top \quad (2)$$

where $\mathbf{U} \in \mathbb{R}^{(d_{in}\times r)}$, $\mathbf{V} \in \mathbb{R}^{(d_{out}\times r)}$, $\boldsymbol{\sigma} \in \mathbb{R}_0^{+r}$. Then, it enforces an additional regularisation term $\mathcal{L}_{SVD}(\Delta\mathbf{W})$ to the loss to imposing orthonormality on the adapter matrices.

$$\Delta\mathbf{W} = \|\mathbf{U}^\top\cdot\mathbf{U}-\mathbf{I}\|_2^2 + \|\mathbf{V}^\top\cdot\mathbf{V}-\mathbf{I}\|_2^2 \quad (3)$$

Despite this constraint allows to interpret the values of $\boldsymbol{\sigma}$ as the eigenvalues and, thus, prune all elements corresponding to null eigenvalues in increases the memory and time requirements of the training process with respect to a normal LoRA.

SoRA builds on top of AdaLoRA, discarding the SVD constraint and substituting the vector of eigenvalues with a gating vector $\mathbf{g} \in \mathbb{R}^r$ and enforcing sparsity adding to the loss a L1 regularisation penalty on g. This simple, yet effective solution, encourages to prune all elements corresponding to a 0 valued element in the gate, as they will be ignored in the computation of the output (exactly as the elements corresponding to a null eigenvalue). The complete formulation of SoRA includes the proximal gradient update using a thresholding function that ensures training stability. This addition is already part of the optimiser we use in our experiments (see Appendix B for further details).

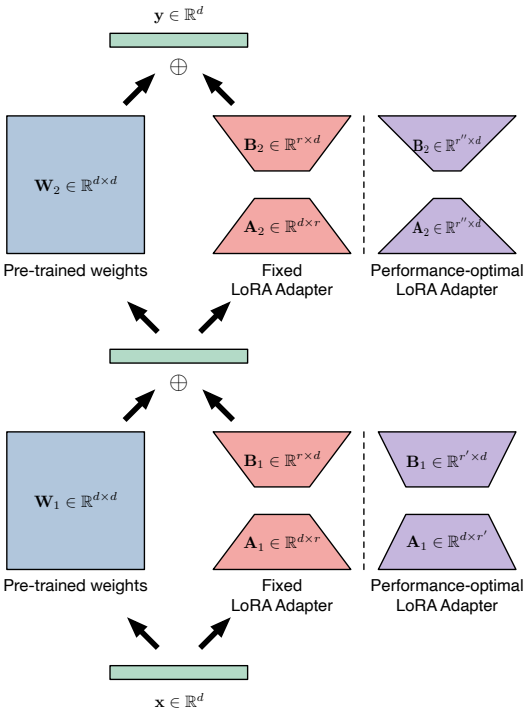


Figure 2: Motivating example: r' and r'' are such that $r'+r''=2r$, so that the total amount of adapters memory is the same with and without optimal allocation.

All the proposed solutions for dynamic rank assignment correctly work to reduce the rank used in the adapter matrices. However, these LoRA variants are limited in the sense that they do not allow for spare (unused) ranks re-assignment and they rather wait for the end of training to prune the matrices. They instead propose starting directly from higher ranks, usually $3r/2$, which increase the overall mem-

ory requirement with respect to a base LoRA operating with the same resources and rank r. Consider the toy example in Figure 2, where we have the comparison between the matrices of LoRA with fixed rank allocation and the matrices with performance-optimal rank allocation. In this case we would have a rank budget of $2r$ that, in the performance-optimal allocation, is divided between r', in the first adapter, and r'', in the second adapter, that $r'+r''=2r$ and $r'>r>r''$. In this configuration, with adapters like AdaLoRA or SoRA, we would need to start at least from a rank budget of $2r'>2r$ to reach the performance optimal allocation, which is above the available budget of $2r$. Moreover, it may be the case where, since we are talking of constrained resources, the model with all the adapters starting from rank r' would not fit in memory.

Besides the theoretical aspects of staying within the rank budget, we also have a "physical" constraint given by the amount of available GPU memory. To tackle this problem we developed the Memory-GELATO tool, which comes as a complement to L1RA. Though accurate estimates of the memory usage we can identify the starting rank without exceeding the available resources. Similarly to other solutions, L1RA can drop the ranks in excess, but differently from the other takes care of re-allocating at runtime those ranks, all of this staying within the given budget.

In this section we described exactly the problems we tackle with our work: *how to get the best performances given a fixed rank or memory budget*? In other words, our contribution is an algorithm that dynamically re-allocates rank amongst adapter matrices in order to maximise performance given a fixed maximum memory budget available, complemented with a tool for memory budget estimation.

4 L1RA

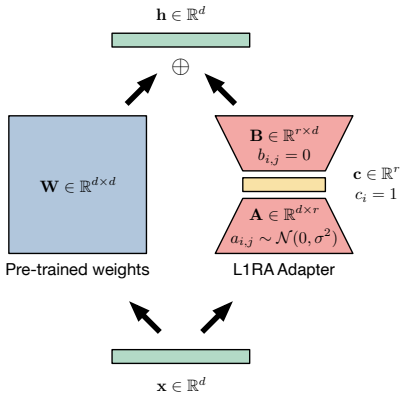


Figure 3: L1RA adapters

L1RA adapters, depicted in Figure 3, extend the LoRA framework by introducing rank pruning and reallocation mechanisms within a fixed rank or memory budget. The goal of L1RA is to identify the performance-optimal rank configuration in computational constrained settings where memory –and time– may be limited. This dynamic rank adjustment ensures that the model efficiently utilises the available resources, enhancing performance without exceeding the same constraints a vanilla LoRA adapter would have.

Mathematically, given an input vector $\mathbf{x} \in \mathbb{R}^{d_{in}}$, we compute the output $\mathbf{h} \in \mathbb{R}^{d_{out}}$ of a L1RA adapter as described in Equation (4). Where $\mathbf{W} \in \mathbb{R}^{d_{in} \times d_{out}}$ is the original matrix of pre-trained weights, $\Delta\mathbf{W} \in \mathbb{R}^{d_{in} \times d_{out}}$ is the adapter matrix of weights decomposed in $\mathbf{A} \in \mathbb{R}^{d_{in} \times r}$, $\mathbf{c} \in \mathbb{R}^r$ and $\mathbf{B} \in \mathbb{R}^{r \times d_{out}}$, and the $\mathrm{diag}(\cdot)$ function outputs a diagonal matrix with the elements of the input vector as values on the diagonal. The rank r depends on the specific adapter and is selected through optimisation during training.

$$\mathbf{h} = \mathbf{x} \cdot (\mathbf{W} + \Delta\mathbf{W}) = \mathbf{x} \cdot \mathbf{W} + \mathbf{x} \cdot (\mathbf{A} \cdot \mathrm{diag}(\mathbf{c}) \cdot \mathbf{B}) \tag{4}$$

The $\mathbf{c}$ vector we introduced, similar to the gating system of SoRA, is a technical device to ease the enforcing of the sparsity constraint. We could have obtained the same effect of imposing sparsity on $\mathbf{c}$ (resulting from the regularisation term $\lambda\|\mathbf{c}\|_1$), by applying the same L_1 constraint to the columns of the input projection matrix $\mathbf{A}$ of a regular LoRA adapter. Doing so would have resulted, however, in much slower redistribution of rank, since an entire column of $\mathbf{A}$ would need to have converged to zero before it could be removed and reassigned to a different matrix, whereas a single component of the $\mathbf{c}$ vector falling to zero is sufficient for reassignment. Thus, while the $\mathbf{c}$ vector does introduce a small number of additional parameters, it results in faster and more direct rank-sparsification, while not affecting the overall transformation of the adapter.

We report the training process of a model using L1RA adapters in the pseudocode detailed in Algorithm 1 and we detail the rank pruning and re-allocation process in Figure 4. The overall training is similar to that of a model using LoRA adapters. The loss function is changed to include the L_1 regularisation term (controlled by the λ hyperparameter) on the elements of the $\mathbf{c}$ vector. Similarly to SoRA, by enforcing sparsity on the $\mathbf{c}$ vector through this regularisation, we achieve rank pruning. In fact, whenever an element of $\mathbf{c}$ is shrunk

Algorithm 1 L1RA pseudocode

Require:
- ϑ ▷ Model parameters
- $\mathcal{D}$ ▷ Data
- $r \in \mathbf{N}^+$ ▷ Initial adapters rank

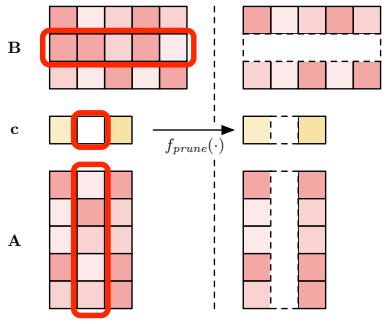
$\Delta\vartheta \leftarrow \{\}$ ▷ Adapter parameters
for $\mathbf{W} \in \vartheta$ **do** ▷ Initialise adapters of all layers
  $\mathbf{A} \leftarrow \mathbf{A} \in \mathbb{R}^{d \times r} \sim \mathcal{N}(0, \sigma^2)$
  $\mathbf{B} \leftarrow \mathbf{0} \in \{0\}^{r \times d}$
  $\mathbf{c} \leftarrow \mathbf{1} \in \{1\}^{r}$
  $\Delta\vartheta \leftarrow \Delta\vartheta \cup \{(\mathbf{A}, \mathbf{B}, \mathbf{c})\}$
end for
for $i \in [0, n_{epochs}) \subseteq \mathbb{N}$ **do** ▷ Iterate over epochs
  for $X \in \mathcal{D}$ **do** ▷ Iterate over training samples
    $\mathcal{L}(\Delta\vartheta) \leftarrow -\ln P(X; \vartheta, \Delta\vartheta) + \lambda \cdot \sum_{(\mathbf{A},\mathbf{B},\mathbf{c}) \in \Delta\vartheta} \|\mathbf{c}\|_1$ ▷ Get loss
    $\Delta\vartheta \leftarrow \Delta\vartheta - \eta \cdot \nabla_{\Delta\vartheta} \cdot \mathcal{L}(\Delta\vartheta)$ ▷ Update adapter weights
    $\rho \leftarrow 0$ ▷ Initialise spare ranks
    $\Delta\vartheta_u \leftarrow []$ ▷ Initialise list of unpruned adapters
    for $(\mathbf{A}, \mathbf{B}, \mathbf{c}) \in \Delta\vartheta$ **do** ▷ Iterate over adapters
      if $\exists c \in \mathbf{c} | c = 0$ **then** ▷ Check for a rank decrease
        $\rho \leftarrow \rho + \sum_{c \in \mathbf{c}} \mathcal{I}(c = 0)$ ▷ Count spare ranks
        $(\mathbf{A}, \mathbf{B}, \mathbf{c}) \leftarrow f_{prune}(\mathbf{A}, \mathbf{B}, \mathbf{c})$ ▷ Apply pruning
      else ▷ Else if not pruned
        $f_{insert}(\Delta\vartheta_u, (\mathbf{A}, \mathbf{B}, \mathbf{c}))$ ▷ Save adapter for reallocation
      end if
    end for
    while $\rho > 0$ **do** ▷ While there are spare ranks
      for $(\mathbf{A}, \mathbf{B}, \mathbf{c}) \in \Delta\vartheta_u$ **do** ▷ Iterate over unpruned adapters
        if $\rho > 0$ **then** ▷ if there are spare ranks
          $(\mathbf{A}, \mathbf{B}, \mathbf{c}) \leftarrow f_{reallocate}(\mathbf{A}, \mathbf{B}, \mathbf{c})$ ▷ Re-allocate a rank
          $\mathbf{c} \leftarrow \mathbf{c} / \sum_{c \in \mathbf{c}} c$ ▷ Normalise $\mathbf{c}$ vector
          $\rho \leftarrow \rho - 1$ ▷ Update spare ranks
        end if
      end for
    end while
  end for
end for
return $\Delta\vartheta$

to 0 the corresponding column in $\mathbf{A}$ and row in $\mathbf{B}$ –the other matrices of the adapter– can be dropped (this is the role of the $f_{prune}(\cdot)$ function).

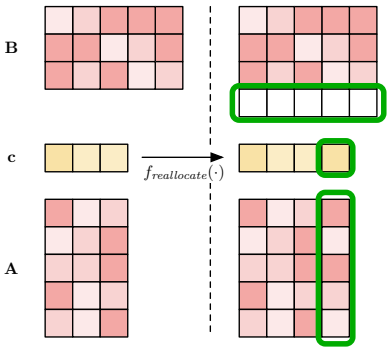
All the spare ranks generated by this pruning process can be re-allocated to the other, unpruned, adapters. Whenever spare ranks are available, the algorithm cycles over the unpruned adapters *sorted by decreasing order of the minimum value in the* $\mathbf{c}$ *vector*, so that

$$(\mathbf{A}_i, \mathbf{B}_i, \mathbf{c}_i) > (\mathbf{A}_j, \mathbf{B}_j, \mathbf{c}_j) \iff \min \mathbf{c}_i > \min \mathbf{c}_j \tag{5}$$

and re-assigns a rank to each adapter until spare ranks are no longer available. In other words, each available additional rank is always redistributed to the particular adapter which is most in need of the rank increase, because its current rank-budget



(a) When a null component in the $\mathbf{c}$ vector of an adapter is detected, the corresponding elements of the adapter are removed using the $f_{prune}(\cdot)$ function, generating a spare rank that will be reallocate.



(b) When a spare rank is available and a needs to be reallocated, the elements on the target adapter are extended by the $f_{reallocate}(\cdot)$ function (values are initialised as in a regular initialisation).

Figure 4: L1RA pruning and reallocation (lighter colours are for lower absolute values, white is 0).

is in full use, with the various components of the $\mathbf{c}$ vector furthest from zero.

The ordering of unpruned adapters is performed in Algorithm 1 by the $f_{insert}(\cdot)$ function when saving them in $\Delta\vartheta_u$. After the rank re-allocation step, the training procedure reprises. This sorting step is inspired by SVD: assuming that in high-dimensional space the matrices $\mathbf{A}$ and $\mathbf{B}$ can be treated as orthogonal and the $\mathbf{c}$ vector mimics the diagonal of the singular value matrix.

Compared to other dynamic rank adapters like AdaLoRA, SoRA and DyLoRA, L1RA offers significant advantages. If we consider a model using vanilla LoRA adapters with a given rank r, since all these other techniques do not account for spare ranks re-allocation, they would require starting from a higher rank initialisation to have the adapters requiring a rank $r' > r$ reach that value, implicitly requiring more memory than the original LoRA would have used. In contrast, L1RA basically maintains almost the same memory usage by reallocating ranks within the fixed budget (as we detail better in

Section 8, it cannot always be the same due to some weight matrices having d_{in} or d_{out} different from others). Additionally, AdaLoRA increases the requirements on time and memory by imposing SVD behavior to the elements of the adapter through additional terms in the loss function. L1RA's approach avoids these additional constraints, ensuring computational efficiency while achieving performance-optimal rank configuration and maintaining memory limits. This makes L1RA a better choice for resource-constrained environments, offering a balanced solution for dynamic rank adaptation.

5 Memory-GELATO

The Memory-GELATO tool is crucial to reach full memory exploitation. In fact, it provides an accurate estimate of the memory required to train a model We identified the following contribution to memory estimates:

- *Model parameters*, which include the weights of all layers and the adapters and is influenced by the numeric precision and quantisation;
- *Steady state memory*, that is all memory reserved to keep track of the intermediate states generated by passing data through the model, the gradients and the optimiser state;
- *Activation*, that is the additional memory used to memorise the activations for gradient checkpointing (reducing the memory footprint of gradients);
- *Loss*, which includes the output logits and the memory used to compute the negative log-likelihood;
- *Other contributions*, which includes all the additional elements increasing memory, like operations at the end of the forward pass and the beginning of the backward pass.

To assess the goodness of Memory-GELATO estimates, we compared the predicted and real values of memory peak usage for different models, different maximum sequence lengths and different batch sizes. In Figure 5, we can see the difference between the estimates and the real values; while, in Table 1, we report quantitative metrics on estimates goodness. Overall, the error in estimated peak memory usage differs from the real one of a few hundreds MBs, including the overestimate we introduced for safety.

Table 1: Memory-GELATO performance in predicting peak memory usage (MAE: Mean Absolute Error, ρ: Spearman correlation coefficient, r Pearson correlation coefficient).

Model	MAE [MB]	ρ	r
Mistral 7B v0.3	203.05	1.0000	0.9998
LLaMa2 7B	109.80	1.0000	0.9999
LLaMa 3.1 8B	159.01	1.0000	0.9999
Phi-3 Mini 4K	146.03	1.0000	0.9998

6 Evaluation

To evaluate L1RA against other adapter approaches, we applied it to fine-tune a LLM in a realistic use case: assistant fine-tuning. Moreover, to demonstrate empirically the practical advantages of L1RA against other approaches we used Memory-GELATO to configure the experiment to maximise memory utilisation. We detail the experimental settings in Appendix B.

We experimented fine-tuning to different LLMs (namely Mistral 7B v0.3 (Jiang et al., 2023) and LLaMa 3.1 8B (Dubey et al., 2024), both quantised at 4 bits precision) to make sure that L1RA is agnostic of the LLM. We selected the OpenOrca data set (Mukherjee et al., 2023)[1] for this assistant fine-tuning.

In this experiment, we compared the test performance and resource consumption of L1RA against LoRA, AdaLoRA. We compared against two versions of AdaLoRA: one targeting the same average rank as LoRA and L1RA starting from an higher rank (1.5 times that of LoRA as suggested in the AdaLoRA documentation), and another version starting from the same rank of LoRA and L1RA and targeting a smaller rank (so that the initial one was 1.5 times that of LoRA, again, as suggested in the AdaLoRA documentation). This fine-tuning task was chosen to demonstrate the practical application of L1RA in efficient fine-tuning of LLMs, particularly in scenarios where fine-tuning on consumer-level GPUs is challenging (e.g., when we reach the limit of usable memory).

Throughout the experiment, we kept track of ranks evolution to analyse the final distribution at the end of training. It this way we can get a better understanding of which components within the Transformer architecture need a more precise

[1]Data set card: https://huggingface.co/datasets/Open-Orca/OpenOrca

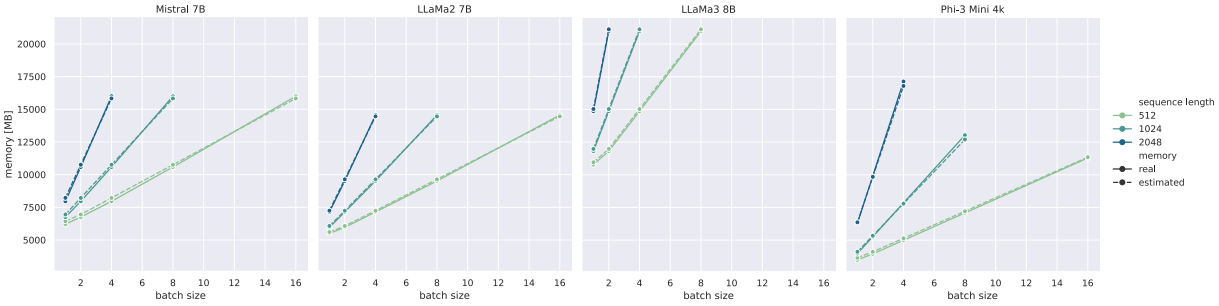
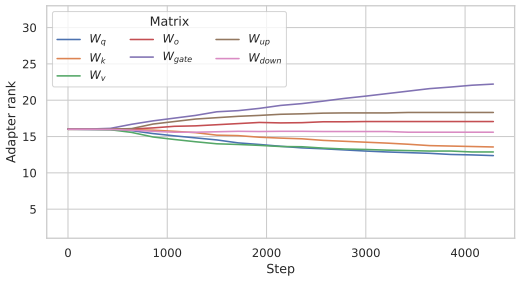


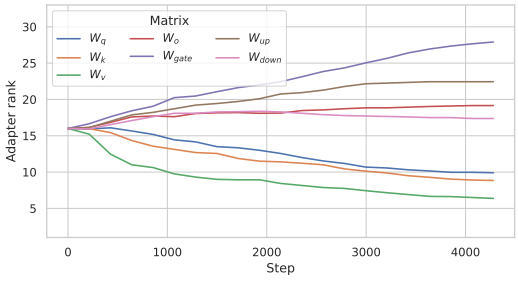
Figure 5: Comparison of peak memory usage estimates from MEMORY-GELATO against actual peak memory usage during training with LORA adapters.

adaptation (identified as those with a higher adapter rank) and shed lights on the internal mechanisms of the Transformer architecture.

7 Results



(a) LLAMA 3.1 8B.



(b) MISTRAL 7B V0.3.

Figure 6: Matrix-wise evolution of layer-wise average L1RA adapters rank during training.

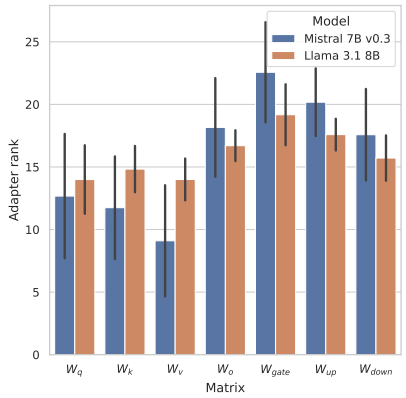


Figure 7: Matrix-wise distribution of layer-wise average L1RA adapters rank at the end of training (error bars show standard deviation).

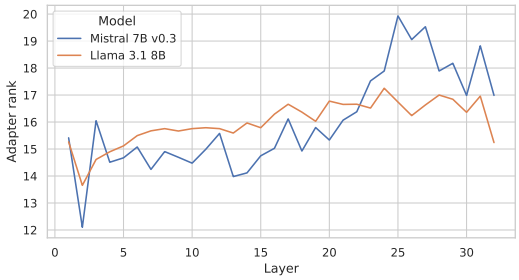


Figure 8: Layer-wise evolution of matrix-wise average L1RA adapters rank at the end of training.

We report the main results of the experiments in Table 2, while the relative values, to ease the comparison, are in Table 3. L1RA achieves the lowest absolute perplexity (PPL) score, improving over both LORA and ADALORA. Moreover, L1RA achieves also the closest training time to that of LORA, with less than 1% difference from LORA. Memory consumption, on the other hand, seems to be similar among the three approaches (most differences from LORA are below 2%) with ADALORA performing better than L1RA (and even LORA in one configuration). The number of adapter (trainable) parameters shows how ADALORA not applying an actual pruning the matrices and requiring an higher starting rank to target the same average ranks of LORA and L1RA increases significantly the number of parameters (50%) without an improvement on the PPL. On the other side, L1RA exchanging freely parameters between matrices of different sizes causes an increase in the number of parameters as training continues; however, the increase is smaller than that of ADALORA and achieves better PPL than both LORA and ADALORA. We discuss

Table 2: Results and resources consuption of chatbot assistant fine-tuning on the OPENORCA data set (*Rank*: is the initial adapters rank –for ADALORA we have the initial rank and target average–, *PPL*: perplexity –lower is better, bold values are the best result for each model–, *Time*: total time for training, validation and testing; *Memory*: peak VRAM usage during training; *No. of adapter parameters*: trainable adapter parameters at the end of training).

Model	Approach	Rank	PPL ↓	Training time [s] ↓	Memory [GB][1]↓	No. of adapter parameters [M] ↓	
						Start of training	End of training
LLAMA 3 8B	LORA	16	3.32	30994.89	13.84	41.94	41.94
	ADALORA	24→16	3.63[2]	32980.28	14.00	62.92	62.92
	ADALORA	16→12	3.57[2]	32964.14	13.76	41.95	41.95
	L1RA	16	**3.25**	31246.40	14.23	41.95	45.16
MISTRAL 7B V0.3	LORA	16	2.93	37891.88	13.58	41.94	41.94
	ADALORA	24→16	3.16[2]	40234.87	13.82	62.92	62.92
	ADALORA	16→12	3.16[2]	40215.02	13.59	41.95	41.95
	L1RA	16	**2.91**	37968.91	13.94	41.95	50.06

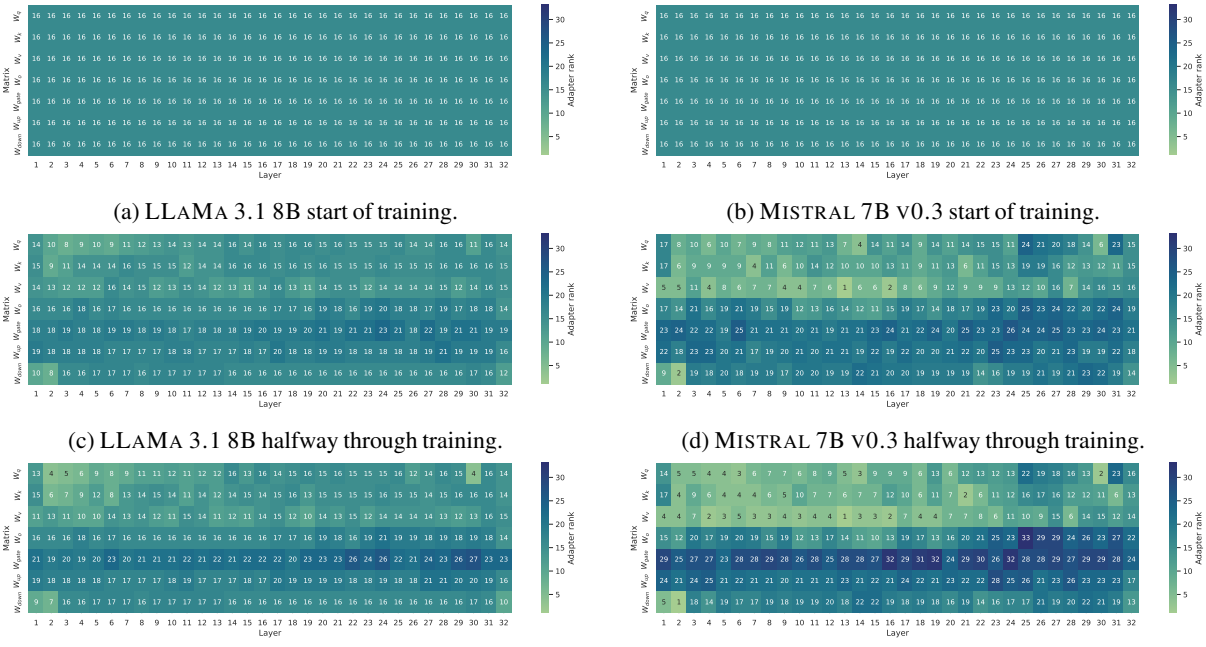
[1] Values measured using PYTORCH utility for measuring GPU device memory usage: https://pytorch.org/docs/stable/generated/torch.cuda.max_memory_allocated.html.

[2] Values are slightly altered because PPL was computed from the loss of the model which included also the regularisation term, separate computations showed that ADALORA PPL was higher than that of LORA and L1RA.

Table 3: Relative results and resources consumption from Table 2 normalised to the LORA fine-tuning.

Model	Approach	Rank	Δ PPL [%] ↓	Δ Training time [%] ↓	Δ Memory [%] ↓	Δ No. of adapter parameters [%][1]↓
LLAMA 3 8B	ADALORA	24→16	9.34	6.41	1.16	50.02
	ADALORA	16→12	7.53	6.35	−0.58	0.02
	L1RA	16	−2.11	0.81	2.82	7.68
MISTRAL 7B V0.3	ADALORA	24→16	7.85	6.18	1.77	50.02
	ADALORA	16→12	7.85	6.13	0.07	0.02
	L1RA	16	−0.68	0.20	2.65	19.36

[1] Values computed on end-of-training parameters.



(a) LLAMA 3.1 8B start of training.

(b) MISTRAL 7B V0.3 start of training.

(c) LLAMA 3.1 8B halfway through training.

(d) MISTRAL 7B V0.3 halfway through training.

(e) LLAMA 3.1 8B end of training.

(f) MISTRAL 7B V0.3 end of training.

Figure 9: Layer-wise and Matrix-wise evolution of L1RA adapters rank during training.

better about memory consumption and number of adapter parameters in Section 8.

In Figure 6 we can the average evolution of the ranks organised per matrix of the Transformer architecture. As we can see, LLAMA and MISTRAL have the same trends: matrices coming from the

Feed-Forward Neural Network (FFNN) layer of the Transformer architecture (up-projection $\mathbf{W}_{up}$, gate $\mathbf{W}_{gate}$, and down-projection $\mathbf{W}_{down}$) are more "rank hungry" than those of the *multi-head self-attention* layers (key $\mathbf{W}_k$, value $\mathbf{W}_v$, query $\mathbf{W}_q$, and output $\mathbf{W}_o$). At the end of training, the difference is clear across all layers, as shown by the averaged rank counts in Figure 7. From Figure 8 we can see another common trend between the two models: ranks are higher in the layers closer to the output of the neural network.

Finally, to report on the individual ranks of each matrix in the Transformer stack, we can see in Figure 9 ranks distributions at the beginning, halfway through and at the end of training. The darker area emerging at the bottom corresponds to the the matrices of the FFNN. We can see how the "rank mass" is higher in these layers especially toward the top of the Transformer network (bottom-right side on the plot) and how it is lower for the multi-head self-attention layer matrices at the bottom of the Transformer network. Moreover, we can wee how with LLAMA this trend is emerging slower: the matrix showing rank distribution at the end is closer to that of MISTRAL halfway through training. Given the higher PPL, we can assume that LLAMA could have been trained for more iterations.

8 Discussion

Values of memory consumption does not comply with our expectations, especially if compared taking into account the rank distributions and the number of trainable parameters. Considering the average ranks and the total number of parameters, we expected to see ADALORA starting from the higher rank having the highest memory consumption and ADALORA starting from the same rank as LORA still consume more memory due to the additional operations to compute the regularisation term, while the memory is even lower in the case of LLAMA. We suspect this is due to some internal optimisation or offloading of the trainer in the HUGGINGFACE's TRANSFORMERS library we used (Wolf et al., 2020). Despite we were not able to locate the source of this difference, we conducted a small experiments on the same data using the same handmade training loop with all adapters and we measured an overall higher memory consumption that was more in line with the number of parameters and the compared techniques. In the next iteration of L1RA we plan to drop the trainer to have more reliable estimates.

To comment on the difference in number of parameters between L1RA and LORA, we can see that despite L1RA not exceeding the rank budget, the amount of parameters (and used memory) is slightly higher than LORA. This is a result of allocating the spare ranks to other adapters working on matrices of different sizes. In particular, as we saw from Figure 7, many ranks are allocated to the feed-forward layers, which have a higher ($4\times$) inner projection dimensionality. Despite this situation, L1RA still achieves a lower resources utilisation when compared to ADALORA.

Finally, to comment on the trends observed in Figures 6 to 9, we can say that trends hint how the FFNN layers at the top of the Transformer stack are contributing more to the task being solved. The high-level features processed in that part of the Transformer network need more precise refinement thus the higher rank. Similarly, we believe that exploiting different information from other tokens in the context is not as important as extracting more refined patterns with the non-linear transformations of the FFNN to have the LLM behave as a chatbot assistant, thus the higher ranks in FFNN layers. This observation agrees with intuition that the higher layers of the network should contribute the most to adapting the network to a specific domain, and that the output and FFNN layers are crucial for storing domain-specific information (as noted by (Geva et al., 2021; Biderman et al., 2023)) that likely needs to be updated by the adapters.

9 Conclusion

In this paper, we introduced L1RA, a novel technique for efficient LLM fine-tuning. By effectively exploiting the dynamic rank assignment given by L_1 regularisation and re-assigning the spare ranks within the available budget, L1RA represents a significant advancement in efficient fine-tuning, offering a promising solution for resource-constrained environments. We completed L1RA with MEMORY-GELATO our tool for GPU memory estimation we can exploit to determine the memory –and thus rank– budget. At this moment we foresee two possible, complementary, directions in the further development of L1RA: we are interested in studying the rank distribution across different models and at a different scales or number of parameter and data-set sizes and we are interested in better understanding the convergence of the proposed method.

Limitations

In this paper, we mainly focused on the development of L1RA for efficient fine-tuning and its evaluation on realistic use cases, rather than exhaustive experiments. The first limitation is in the choice of the LLM: as for now, we evaluated the results using only MISTRAL 7B V0.3 and LLAMA 3 8B. A proper evaluation would require exploring other openly accessible models of the same and different sizes that would fit on a consumer-level GPU. The second limitation is the choice of the evaluation data set: we considered only the task of instruction fine-tuning since it is a common use case and since it covers many tasks an LLM is required to solve, however a more extensive evaluation exploring different tasks would improve the understanding of L1RA's capabilities.

Ethics Statement

The authors do not foresee any considerable risks associated with the work presented in this paper. In principle, the L1RA algorithm is intended to make fine-tuning of LLMs more efficient and the MEMORY-GELATO tool is intended for estimating memory consumption such fine-tunings. The authors made the source code publicly available to ensure the reproducibility of the experiments. Refer to Appendix A for further details.

Acknowledgements

This work was partially supported by the FAIR (Future Artificial Intelligence Research) project, funded by the NextGenerationEU program within the PNRR-PE-AI scheme (M4C2, Investment 1.3, Line on Artificial Intelligence), by funding from the pilot program Core Informatics at KIT (KiKIT) of the Helmholtz Association (HGF), and supported by the German Research Foundation (DFG) - SFB 1608 - 501798263 and KASTEL Security Research Labs, Karlsruhe.

References

Ankur Bapna and Orhan Firat. 2019. Simple, scalable adaptation for neural machine translation. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing, EMNLP-IJCNLP 2019, Hong Kong, China, November 3-7, 2019*, pages 1538–1548. Association for Computational Linguistics.

Stella Biderman, USVSN Sai Prashanth, Lintang Sutawika, Hailey Schoelkopf, Quentin Anthony, Shivanshu Purohit, and Edward Raff. 2023. Emergent and predictable memorization in large language models. In *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*.

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language models are few-shot learners. In *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*.

Tri Dao. 2024. Flashattention-2: Faster attention with better parallelism and work partitioning. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net.

Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. 2023. Qlora: Efficient finetuning of quantized llms. In *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*.

Ning Ding, Xingtai Lv, Qiaosen Wang, Yulin Chen, Bowen Zhou, Zhiyuan Liu, and Maosong Sun. 2023. Sparse low-rank adaptation of pre-trained language models. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing, EMNLP 2023, Singapore, December 6-10, 2023*, pages 4133–4145. 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, Anirudh Goyal, Anthony Hartshorn, Aobo Yang, Archi Mitra, Archie Sravankumar, Artem Korenev, Arthur Hinsvark, Arun Rao, Aston Zhang, Aurelien Rodriguez, Austen Gregerson, Ava Spataru, Baptiste Roziere, Bethany Biron, Binh Tang, Bobbie Chern, Charlotte Caucheteux, Chaya Nayak, Chloe Bi, Chris Marra, Chris McConnell, Christian Keller, Christophe Touret, Chunyang Wu, Corinne Wong, Cristian Canton Ferrer, Cyrus Nikolaidis, Damien Allonsius, Daniel Song, Danielle Pintz, Danny Livshits, David Esiobu, Dhruv Choudhary, Dhruv Mahajan, Diego Garcia-Olano, Diego Perino, Dieuwke Hupkes, Egor Lakomkin, Ehab AlBadawy, Elina Lobanova, Emily Dinan, Eric Michael Smith, Filip Radenovic, Frank Zhang, Gabriel Synnaeve, Gabrielle Lee, Georgia Lewis Anderson, Graeme Nail, Gregoire Mialon,

Guan Pang, Guillem Cucurell, Hailey Nguyen, Hannah Korevaar, Hu Xu, Hugo Touvron, Iliyan Zarov, Imanol Arrieta Ibarra, Isabel Kloumann, Ishan Misra, Ivan Evtimov, Jade Copet, Jaewon Lee, Jan Geffert, Jana Vranes, Jason Park, Jay Mahadeokar, Jeet Shah, Jelmer van der Linde, Jennifer Billock, Jenny Hong, Jenya Lee, Jeremy Fu, Jianfeng Chi, Jianyu Huang, Jiawen Liu, Jie Wang, Jiecao Yu, Joanna Bitton, Joe Spisak, Jongsoo Park, Joseph Rocca, Joshua Johnstun, Joshua Saxe, Junteng Jia, Kalyan Vasuden Alwala, Kartikeya Upasani, Kate Plawiak, Ke Li, Kenneth Heafield, Kevin Stone, Khalid El-Arini, Krithika Iyer, Kshitiz Malik, Kuenley Chiu, Kunal Bhalla, Lauren Rantala-Yeary, Laurens van der Maaten, Lawrence Chen, Liang Tan, Liz Jenkins, Louis Martin, Lovish Madaan, Lubo Malo, Lukas Blecher, Lukas Landzaat, Luke de Oliveira, Madeline Muzzi, Mahesh Pasupuleti, Mannat Singh, Manohar Paluri, Marcin Kardas, Mathew Oldham, Mathieu Rita, Maya Pavlova, Melanie Kambadur, Mike Lewis, Min Si, Mitesh Kumar Singh, Mona Hassan, Naman Goyal, Narjes Torabi, Nikolay Bashlykov, Nikolay Bogoychev, Niladri Chatterji, Olivier Duchenne, Onur Çelebi, Patrick Alrassy, Pengchuan Zhang, Pengwei Li, Petar Vasic, Peter Weng, Prajjwal Bhargava, Pratik Dubal, Praveen Krishnan, Punit Singh Koura, Puxin Xu, Qing He, Qingxiao Dong, Ragavan Srinivasan, Raj Ganapathy, Ramon Calderer, Ricardo Silveira Cabral, Robert Stojnic, Roberta Raileanu, Rohit Girdhar, Rohit Patel, Romain Sauvestre, Ronnie Polidoro, Roshan Sumbaly, Ross Taylor, Ruan Silva, Rui Hou, Rui Wang, Saghar Hosseini, Sahana Chennabasappa, Sanjay Singh, Sean Bell, Seohyun Sonia Kim, Sergey Edunov, Shaoliang Nie, Sharan Narang, Sharath Raparthy, Sheng Shen, Shengye Wan, Shruti Bhosale, Shun Zhang, Simon Vandenhende, Soumya Batra, Spencer Whitman, Sten Sootla, Stephane Collot, Suchin Gururangan, Sydney Borodinsky, Tamar Herman, Tara Fowler, Tarek Sheasha, Thomas Georgiou, Thomas Scialom, Tobias Speckbacher, Todor Mihaylov, Tong Xiao, Ujjwal Karn, Vedanuj Goswami, Vibhor Gupta, Vignesh Ramanathan, Viktor Kerkez, Vincent Gonguet, Virginie Do, Vish Vogeti, Vladan Petrovic, Weiwei Chu, Wenhan Xiong, Wenyin Fu, Whitney Meers, Xavier Martinet, Xiaodong Wang, Xiaoqing Ellen Tan, Xinfeng Xie, Xuchao Jia, Xuewei Wang, Yaelle Goldschlag, Yashesh Gaur, Yasmine Babaei, Yi Wen, Yiwen Song, Yuchen Zhang, Yue Li, Yuning Mao, Zacharie Delpierre Coudert, Zheng Yan, Zhengxing Chen, Zoe Papakipos, Aaditya Singh, Aaron Grattafiori, Abha Jain, Adam Kelsey, Adam Shajnfeld, Adithya Gangidi, Adolfo Victoria, Ahuva Goldstand, Ajay Menon, Ajay Sharma, Alex Boesenberg, Alex Vaughan, Alexei Baevski, Allie Feinstein, Amanda Kallet, Amit Sangani, Anam Yunus, Andrei Lupu, Andres Alvarado, Andrew Caples, Andrew Gu, Andrew Ho, Andrew Poulton, Andrew Ryan, Ankit Ramchandani, Annie Franco, Aparajita Saraf, Arkabandhu Chowdhury, Ashley Gabriel, Ashwin Bharambe, Assaf Eisenman, Azadeh Yazdan, Beau James, Ben Maurer, Benjamin Leonhardi, Bernie Huang, Beth Loyd, Beto De Paola, Bhargavi Paranjape, Bing Liu, Bo Wu, Boyu Ni, Braden Hancock, Bram Wasti, Brandon Spence, Brani Stojkovic, Brian Gamido, Britt Montalvo, Carl Parker, Carly Burton, Catalina Mejia, Changhan Wang, Changkyu Kim, Chao Zhou, Chester Hu, Ching-Hsiang Chu, Chris Cai, Chris Tindal, Christoph Feichtenhofer, Damon Civin, Dana Beaty, Daniel Kreymer, Daniel Li, Danny Wyatt, David Adkins, David Xu, Davide Testuggine, Delia David, Devi Parikh, Diana Liskovich, Didem Foss, Dingkang Wang, Duc Le, Dustin Holland, Edward Dowling, Eissa Jamil, Elaine Montgomery, Eleonora Presani, Emily Hahn, Emily Wood, Erik Brinkman, Esteban Arcaute, Evan Dunbar, Evan Smothers, Fei Sun, Felix Kreuk, Feng Tian, Firat Ozgenel, Francesco Caggioni, Francisco Guzmán, Frank Kanayet, Frank Seide, Gabriela Medina Florez, Gabriella Schwarz, Gada Badeer, Georgia Swee, Gil Halpern, Govind Thattai, Grant Herman, Grigory Sizov, Guangyi, Zhang, Guna Lakshminarayanan, Hamid Shojanazeri, Han Zou, Hannah Wang, Hanwen Zha, Haroun Habeeb, Harrison Rudolph, Helen Suk, Henry Aspegren, Hunter Goldman, Igor Molybog, Igor Tufanov, Irina-Elena Veliche, Itai Gat, Jake Weissman, James Geboski, James Kohli, Japhet Asher, Jean-Baptiste Gaya, Jeff Marcus, Jeff Tang, Jennifer Chan, Jenny Zhen, Jeremy Reizenstein, Jeremy Teboul, Jessica Zhong, Jian Jin, Jingyi Yang, Joe Cummings, Jon Carvill, Jon Shepard, Jonathan McPhie, Jonathan Torres, Josh Ginsburg, Junjie Wang, Kai Wu, Kam Hou U, Karan Saxena, Karthik Prasad, Kartikay Khandelwal, Katayoun Zand, Kathy Matosich, Kaushik Veeraraghavan, Kelly Michelena, Keqian Li, Kun Huang, Kunal Chawla, Kushal Lakhotia, Kyle Huang, Lailin Chen, Lakshya Garg, Lavender A, Leandro Silva, Lee Bell, Lei Zhang, Liangpeng Guo, Licheng Yu, Liron Moshkovich, Luca Wehrstedt, Madian Khabsa, Manav Avalani, Manish Bhatt, Maria Tsimpoukelli, Martynas Mankus, Matan Hasson, Matthew Lennie, Matthias Reso, Maxim Groshev, Maxim Naumov, Maya Lathi, Meghan Keneally, Michael L. Seltzer, Michal Valko, Michelle Restrepo, Mihir Patel, Mik Vyatskov, Mikayel Samvelyan, Mike Clark, Mike Macey, Mike Wang, Miquel Jubert Hermoso, Mo Metanat, Mohammad Rastegari, Munish Bansal, Nandhini Santhanam, Natascha Parks, Natasha White, Navyata Bawa, Nayan Singhal, Nick Egebo, Nicolas Usunier, Nikolay Pavlovich Laptev, Ning Dong, Ning Zhang, Norman Cheng, Oleg Chernoguz, Olivia Hart, Omkar Salpekar, Ozlem Kalinli, Parkin Kent, Parth Parekh, Paul Saab, Pavan Balaji, Pedro Rittner, Philip Bontrager, Pierre Roux, Piotr Dollar, Polina Zvyagina, Prashant Ratanchandani, Pritish Yuvraj, Qian Liang, Rachad Alao, Rachel Rodriguez, Rafi Ayub, Raghotham Murthy, Raghu Nayani, Rahul Mitra, Raymond Li, Rebekkah Hogan, Robin Battey, Rocky Wang, Rohan Maheswari, Russ Howes, Ruty Rinott, Sai Jayesh Bondu, Samyak Datta, Sara Chugh, Sara Hunt, Sargun Dhillon, Sasha Sidorov, Satadru Pan, Saurabh Verma, Seiji Yamamoto, Sharadh Ramaswamy, Shaun Lindsay, Shaun Lindsay, Sheng Feng, Shenghao Lin, Shengxin Cindy Zha, Shiva Shankar, Shuqiang Zhang, Shuqiang Zhang, Sinong Wang, Sneha Agarwal, Soji Sajuyigbe, Soumith Chintala, Stephanie Max, Stephen Chen, Steve Kehoe,

Steve Satterfield, Sudarshan Govindaprasad, Sumit Gupta, Sungmin Cho, Sunny Virk, Suraj Subramanian, Sy Choudhury, Sydney Goldman, Tal Remez, Tamar Glaser, Tamara Best, Thilo Kohler, Thomas Robinson, Tianhe Li, Tianjun Zhang, Tim Matthews, Timothy Chou, Tzook Shaked, Varun Vontimitta, Victoria Ajayi, Victoria Montanez, Vijai Mohan, Vinay Satish Kumar, Vishal Mangla, Vlad Ionescu, Vlad Poenaru, Vlad Tiberiu Mihailescu, Vladimir Ivanov, Wei Li, Wenchen Wang, Wenwen Jiang, Wes Bouaziz, Will Constable, Xiaocheng Tang, Xiaofang Wang, Xiaojian Wu, Xiaolan Wang, Xide Xia, Xilun Wu, Xinbo Gao, Yanjun Chen, Ye Hu, Ye Jia, Ye Qi, Yenda Li, Yilin Zhang, Ying Zhang, Yossi Adi, Youngjin Nam, Yu, Wang, Yuchen Hao, Yundi Qian, Yuzi He, Zach Rait, Zachary DeVito, Zef Rosnbrick, Zhaoduo Wen, Zhenyu Yang, and Zhiwei Zhao. 2024. The llama 3 herd of models. *CoRR*, abs/2407.21783.

Mor Geva, Roei Schuster, Jonathan Berant, and Omer Levy. 2021. Transformer feed-forward layers are key-value memories. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing, EMNLP 2021, Virtual Event / Punta Cana, Dominican Republic, 7-11 November, 2021*, pages 5484–5495. Association for Computational Linguistics.

Soufiane Hayou, Nikhil Ghosh, and Bin Yu. 2024. Lora+: Efficient low rank adaptation of large models. *CoRR*, abs/2402.12354.

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 *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*, volume 97 of *Proceedings of Machine Learning Research*, pages 2790–2799. PMLR.

Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2022. Lora: Low-rank adaptation of large language models. In *The Tenth International Conference on Learning Representations, ICLR 2022, Virtual Event, April 25-29, 2022*. OpenReview.net.

Albert Q. Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de Las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, Lélio Renard Lavaud, Marie-Anne Lachaux, Pierre Stock, Teven Le Scao, Thibaut Lavril, Thomas Wang, Timothée Lacroix, and William El Sayed. 2023. Mistral 7b. *CoRR*, abs/2310.06825.

Damjan Kalajdzievski. 2023. A rank stabilization scaling factor for fine-tuning with lora. *CoRR*, abs/2312.03732.

Dawid Jan Kopiczko, Tijmen Blankevoort, and Yuki Markus Asano. 2023. Vera: Vector-based random matrix adaptation. *CoRR*, abs/2310.11454.

Xiang Lisa Li and Percy Liang. 2021. Prefix-tuning: Optimizing continuous prompts for generation. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing, ACL/IJCNLP 2021, (Volume 1: Long Papers), Virtual Event, August 1-6, 2021*, pages 4582–4597. Association for Computational Linguistics.

Ilya Loshchilov and Frank Hutter. 2019. Decoupled weight decay regularization. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net.

Sadhika Malladi, Tianyu Gao, Eshaan Nichani, Alex Damian, Jason D. Lee, Danqi Chen, and Sanjeev Arora. 2023. Fine-tuning language models with just forward passes. In *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*.

Subhabrata Mukherjee, Arindam Mitra, Ganesh Jawahar, Sahaj Agarwal, Hamid Palangi, and Ahmed Awadallah. 2023. Orca: Progressive learning from complex explanation traces of GPT-4. *CoRR*, abs/2306.02707.

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

Victor Sanh, Albert Webson, Colin Raffel, Stephen Bach, Lintang Sutawika, Zaid Alyafeai, Antoine Chaffin, Arnaud Stiegler, Arun Raja, Manan Dey, M Saiful Bari, Canwen Xu, Urmish Thakker, Shanya Sharma Sharma, Eliza Szczechla, Taewoon Kim, Gunjan Chhablani, Nihal V. Nayak, Debajyoti Datta, Jonathan Chang, Mike Tian-Jian Jiang, Han Wang, Matteo Manica, Sheng Shen, Zheng Xin Yong, Harshit Pandey, Rachel Bawden, Thomas Wang, Trishala Neeraj, Jos Rozen, Abheesht Sharma, Andrea Santilli, Thibault Févry, Jason Alan Fries, Ryan Teehan, Teven Le Scao, Stella Biderman, Leo Gao, Thomas Wolf, and Alexander M. Rush. 2022. Multitask prompted training enables zero-shot task generalization. In *The Tenth International Conference on Learning Representations, ICLR 2022, Virtual Event, April 25-29, 2022*. OpenReview.net.

Vincenzo Scotti, Licia Sbattella, and Roberto Tedesco. 2024. A primer on seq2seq models for generative chatbots. *ACM Comput. Surv.*, 56(3):75:1–75:58.

Mojtaba Valipour, Mehdi Rezagholizadeh, Ivan Kobyzev, and Ali Ghodsi. 2023. Dylora: Parameter-efficient tuning of pre-trained models using dynamic search-free low-rank adaptation. In *Proceedings of the 17th Conference of the European Chapter of the Association for Computational Linguistics, EACL 2023, Dubrovnik, Croatia, May 2-6, 2023*, pages 3266–3279. Association for Computational Linguistics.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi 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 M. 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, EMNLP 2020 - Demos, Online, November 16-20, 2020*, pages 38–45. Association for Computational Linguistics.

Qingru Zhang, Minshuo Chen, Alexander Bukharin, Pengcheng He, Yu Cheng, Weizhu Chen, and Tuo Zhao. 2023. Adaptive budget allocation for parameter-efficient fine-tuning. In *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net.

Wayne Xin Zhao, Kun Zhou, Junyi Li, Tianyi Tang, Xiaolei Wang, Yupeng Hou, Yingqian Min, Beichen Zhang, Junjie Zhang, Zican Dong, Yifan Du, Chen Yang, Yushuo Chen, Zhipeng Chen, Jinhao Jiang, Ruiyang Ren, Yifan Li, Xinyu Tang, Zikang Liu, Peiyu Liu, Jian-Yun Nie, and Ji-Rong Wen. 2023. A survey of large language models. *CoRR*, abs/2303.18223.

A Source Code Availability

We share the source code associated with this paper for full reproducibility and transparency. All the source material to replicate the experiments is available on GitHub:

- L1RA: https://github.com/raul-singh/L1RA/tree/dev-exp;
- MEMORY-GELATO: https://github.com/raul-singh/memory-gelato.

B Evaluation setup

In this section, we provide the hyperparamers we used for in experimental evaluations to ensure full reproducibility. We report the hyperparameters we used with the OPENORCA data set in Table 4. In the table, we use the following notation:

- r is the initial rank of L1RA, LORA and ADALORA adapters;
- α is the scaling of L1RA, LORA and ADALORA;
- $p_{dropout}$ is the dropout probability of L1RA, LORA and ADALORA;
- η is the learning rate;
- λ is the regularisation coefficient or L1RA or ADALORA;

One important detail of our experiments is the choice of the optimiser, we implemented a variant of *AdamW* (Loshchilov and Hutter, 2019) (which is the most common optimiser for LLMs), to support decoupled regularisation for both L_1 and L_2 regularisations. We refer to this variant as *AdamE*, where the "E" refers to *ElasticNet*: the combined L_1 and L_2 regulariser[2]. The addition is the decoupled L_1 regularisation that avoids the update of the lasso constraint being scaled by the adaptive learning rate and momentum hyperparameters. This scaling affects negatively the shrinking of the parameters, showing it down.

Since we apply learning rate warm-up and cosine scheduling to shrink η to zero, we find useful keep a separate constant learning rate for the parameters in the $\mathbf{c}$ vectors. To avoid introducing unnecessary hyperaparameters we use the same η of the rest of the parameters, but whithout warm-up and scheduling.

[2]AdamE implementation https://github.com/vincenzo-scotti/bitsandbytes/tree/dev-adame

Table 4: OPENORCA hyperparameters.

Model	Hyperparameter	Value
LLAMA 3.1 8B	Max. sequence length	1024 tokens
	r	16
	α	16
	$p_{dropout}$	10^{-1}
	Compute d-type	`bfloat16`
	Attn. implementation	Flash attn. 2 (Dao, 2024)
	Optimiser	AdamE (paged, 32 bit)
	η	10^{-4}
	η schedule	cosine
	η warm-up ratio	5%
	Max grad. norm	1
	Epochs	1
	Batch size	4
	Accum. steps	4
	λ_{L1RA}	10^{-3}
	$\eta_{\mathbf{c}}$ (L1RA)	10^{-2}
	Rank update period (L1RA)	5% training steps
	λ_{ADALORA}	10^{-3}
	t_{init} (ADALORA)	5% training steps
	Δt (ADALORA)	5% training steps
MISTRAL 7B V0.3	Max. sequence length	1024 tokens
	r	16
	α	16
	$p_{dropout}$	10^{-1}
	Compute d-type	`bfloat16`
	Attn. implementation	Flash attn. 2 (Dao, 2024)
	Optimiser	AdamE (paged, 32 bit)
	η	10^{-4}
	η schedule	cosine
	η warm-up ratio	5%
	Max grad. norm	1
	Epochs	1
	Batch size	4
	Accum. steps	4
	λ_{L1RA}	10^{-3}
	$\eta_{\mathbf{c}}$ (L1RA)	10^{-2}
	Rank update period (L1RA)	5% training steps
	λ_{ADALORA}	10^{-3}
	t_{init} (ADALORA)	5% training steps
	Δt (ADALORA)	5% training steps

We conducted all experiments on the same machine with the following hardware configuration:

- CPU: Intel Core i9-13900K;
- RAM: 64 GB;
- GPU: NVIDIA GeForce RTX 4090.

We used as much shared parameters across the three approaches we compare (L1RA, LORA and ADALORA) as possible to have a fair comparison.