CLAM: Unifying Finetuning, Quantization, and Pruning by Chaining LLM Adapter Modules

Neelay Velingker [1] **Amish Sethi** [*1] **Jason Liu** [*1] **William Dodds** [*1] **Zhiqiu Xu** [1] **Saikat Dutta** [1] **Mayur Naik** [1] **Eric Wong** [1]

Abstract

As LLMs have grown in size and applicability, so too have the number of methods that adapt them for downstream tasks. Recent works to address challenges in memory consumption, task performance, and inference efficiency have led to the fields of parameter-efficient finetuning (PEFT), quantization, and pruning, among others. While it is useful to combine their benefits, composing these techniques in flexible ways is challenging due to the changes each method makes to the model and any restrictions they might impose. To address these challenges, we develop an algebraic abstraction called CLAM that enables unlimited chaining of popular resource-efficient methods on nearly every modern LLM with minimal overhead. We demonstrate that CLAM can create new compositions of techniques that achieve SOTA performance on specializing compressed models across multiple benchmarks.

1. Introduction

As LLMs continue to scale beyond the capabilities of consumer hardware, there is a growing need to democratize them. Emerging approaches seek to accomplish this objective by combining the benefits of optimizations such as PEFT, quantization, and pruning (Dettmers et al., 2024; Li et al., 2023; Zhang et al., 2023a;b).

This is a nontrivial task in general. PEFT methods often change model architectures with new layer types and operations. Similarly, quantization produces replacement weights having different dimensions, data types, and operator semantics. Pruning yields new weights, some of whose values must remain constant after subsequent operations.

[*]Equal contribution [1]University of Pennsylvania, Philadelphia, USA. Correspondence to: Neelay Velingker <neelay@seas.upenn.edu >.

ICML 2024 workshop on Efficient Systems for Foundation Models, Vienna, Austria. PMLR 235, 2024. Copyright 2024 by the author(s).

In this paper, we seek to address these fundamental challenges and make the following contributions:

Unifying abstraction. To connect PEFT, quantization, and pruning, we reformulate popular methods in these domains according to a new, functionally equivalent algebraic abstraction. Past work in unifying the field of PEFT has viewed their methods as modifications to intermediate activations (Chen et al., 2022; He et al., 2022), and have generally been rigid in the techniques they combine. We take an alternate approach by instead considering all these methods as modifications to the model's weight tensors using a computation graph, allowing us to combine and stack techniques in a novel manner.

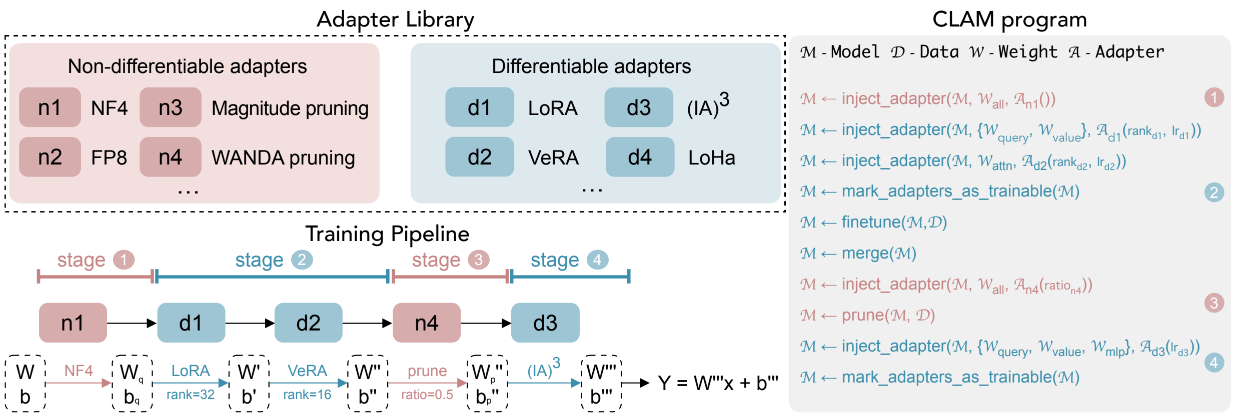
General framework. We implement this abstraction in our new framework **CLAM** which stands for Chaining LLM Adapter Modules. CLAM is compatible with nearly every transformer in Hugging Face's `transformers` library with minimal configuration overhead. It also incorporates many SOTA techniques for making models more accessible, reformulated as adapters that can be combined and chained in an unlimited fashion. This framework yields a novel, more efficient, and performant adapter design space that subsumes prior works at this intersection.

Empirical results. We evaluate CLAM on well-established finetuning benchmarks in GLUE (Wang et al., 2018) and SuperGLUE (Wang et al., 2019) to produce specialized models that minimize model memory consumption while maximizing downstream task performance better than comparable prior methods. Specifically, CLAM's newly discovered PEFT techniques outperform their counterparts by as much as 6.5%. Furthermore, CLAM realizes a novel quantization scheme that not only surpasses the state-of-the-art QLoRA (Dettmers et al., 2024) but also shows comparable performance when further compressed with pruning, effectively halving the number of non-zero bits.

2. Background

2.1. Techniques for Accessible LLMs

Quantization (Dettmers et al., 2022; Frantar et al., 2022) and pruning (Frantar & Alistarh, 2023b; Sun et al., 2023) are two



```
𝓜 - Model  𝓓 - Data  𝓦 - Weight  𝓐 - Adapter

𝓜 ← inject_adapter(𝓜, 𝓦_all, 𝓐_n1())
𝓜 ← inject_adapter(𝓜, {𝓦_query, 𝓦_value}, 𝓐_d1(rank_d1, lr_d1))
𝓜 ← inject_adapter(𝓜, 𝓦_attn, 𝓐_d2(rank_d2, lr_d2))
𝓜 ← mark_adapters_as_trainable(𝓜)
𝓜 ← finetune(𝓜,𝓓)
𝓜 ← merge(𝓜)
𝓜 ← inject_adapter(𝓜, 𝓦_all, 𝓐_n4(ratio_n4))
𝓜 ← prune(𝓜, 𝓓)
𝓜 ← inject_adapter(𝓜, {𝓦_query, 𝓦_value, 𝓦_mlp}, 𝓐_d3(lr_d3))
𝓜 ← mark_adapters_as_trainable(𝓜)
```

Figure 1. **CLAM overview.** CLAM enables unlimited chaining of popular techniques in PEFT, quantization, and pruning on LLMs.

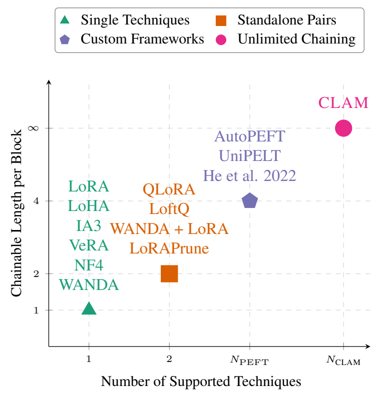


Figure 2. **Space of combinatory LLM techniques.** Each design approach is labeled with exemplar methods. N_{PEFT} is the total number of PEFT techniques. N_{CLAM} is the number of CLAM-compatible techniques that grows over time with new discoveries.

well-studied non-differentiable techniques for compressing LLMs. Quantization (Appendix A.1) discretizes floating-point weights into a lower-bit data type, whereas pruning (Appendix A.2) saves memory via weight sparsification.

Parameter-efficient finetuning (PEFT) aims to match the downstream task performance of full finetuning while training as few parameters as possible. This saves on memory usage and training time. PEFT methods such as LoRA (Hu et al., 2021), (IA)^3 (Liu et al., 2022), LoHA (Hyeon-Woo et al., 2021), and VeRA (Kopiczko et al., 2024) introduce tunable adapters into a given transformer while the original pretrained parameters are frozen. Appendix A.3 discusses each technique in detail and why they are challenging to compose with other techniques.

2.2. Combining Resource-Efficient Techniques

The current space of combining techniques can be divided into two general design patterns as depicted in Figure 2: standalone pairs and custom hybrid frameworks.

Standalone pairs (Dettmers et al., 2024; Sun et al., 2023) select two specific techniques (e.g. NF4 quantization and LoRA) to combine but do not take composability with other techniques as a design goal. Attempting to combine new techniques with these pairs often leads to *interface* errors, where the implementation's limited interoperability manifests as type errors or undefined behavior. Appendix A.4 discusses standalone pairs in more detail.

Custom hybrid frameworks support a mixture of techniques (pre-dominantly PEFT) in an *architecturally* restrictive manner, where the number of active techniques per transformer block is fixed (typically 3–4) and the space of available techniques is often manually pre-determined and non-extensible. In (He et al., 2022), (Mao et al., 2022), and (Zhou et al., 2024), at most 2 distinct techniques can operate on the same weight matrix within a block.

2.3. CLAM Motivation

Figure 1 illustrates a training pipeline that can be encountered when exploring the space of technique combinations. This proposed pipeline highlights some of the challenges when designing a unified framework for chaining techniques, which are detailed in Appendix A.5. In summary, combining any two techniques—the most basic case—involves carefully balancing numerous restrictions and design choices, and the complexity only increases as one attempts to integrate additional techniques. CLAM offers a mathematically and algorithmically-unified solution to this problem.

3. The CLAM Framework

We begin by establishing a common interface over which each optimization must operate. Recall that pruning and quantization directly modify the weights over which they are applied; conversely, PEFT techniques operate as adapters modifying a hidden state. Therefore, we begin by reformulating PEFT adapters to perform weight modification instead, akin to merging adapters with gradient tracking. We similary reformulate non-differentiable methods (e.g. pruning, quantization), and additionally introduce abstractions to manage finetuning before and after applying them.

Table 1. **CLAM-aligned reformulation over a linear layer**

Method	Original Formulation	CLAM-Aligned Formulation
Format	$y = Wx + b$	$y = W'x + b'$
LoRA	$y = Wx + b + cABx$	$y = (W + cAB)x + b$
(IA)^3	$y = (Wx + b) \odot l$	$y = (W diag(l))x + (b \odot l)$
LoHA	$y = Wx + b + c(A_1B_1 \odot A_2B_2)x$	$y = (W + c(A_1B_1 \odot A_2B_2))x + b$
VeRA	$y = Wx + b + \Lambda_A A \Lambda_B Bx$	$y = (W + \Lambda_A A \Lambda_B B)x + b$
Pruning	$y = (M \odot W)x + b$	$y = (M \odot W)x + b$
Quantization	$y = Q(W)x + b$	$y = Q^{-1}(Q(W))x + b$

3.1. Technique Reformulation

Modern libraries and literature design PEFT adapters as modules inserted into a target network to affect change upon intermediate activations. However, this design presents a challenge. By individually modifying the hidden state in a black-box fashion, any two techniques cannot be mathematically combined without understanding their inner mechanics. To resolve this without breaking semantic equivalence, we observe that recent PEFT techniques have been designed to merge any additionally introduced parameters into the original weights to minimize latency overhead otherwise introduced by adding new modules. We leverage this emergent trend, intended to facilitate low-latency inference, to design the adapter interface in CLAM as follows:

$$\mathcal{A}(W, b) \rightarrow (W', b')$$

where W', b' have the same data type and shape as tensors W, b. We then rely on automatic differentiation to actively track and back-propagate to these live merged parameters during finetuning. We posit that any technique that can be reformulated as some $y = W'x + b'$ is fully compatible with CLAM in a modular fashion.

Table 1 shows the reformulation of popular techniques from §2.1 using CLAM's interface. Pruning uses this interface without reformulation as it, by definition, take as input and produce as outputs weights of the same format. Note that for a quantized weight matrix $Q(W)$, we can rely on the dequantization function Q^{-1} to retrieve an approximate matrix W' of the same shape and data type as W. CLAM supports quantization methods that perfectly reconstruct zeros, allowing for pruning structures to stay intact. It is important to acknowledge that a post-training quantization or pruning method may be non-differentiable; that is, for a pruning function $P(W)$ or quantization function $Q(W)$, there may be no computable $\frac{\partial P(W)}{\partial W}$ nor $\frac{\partial Q(W)}{\partial W}$, depending on the method, which we address in §3.2. Given that most modern transformers operate using linear layers, they need only to specify the names of target modules to work with the linear layer interface of CLAM.

3.2. Chaining and Staging

Because all adapters can expect the same type of input and output, CLAM is now able to limitlessly chain differentiable adapters to construct new PEFT techniques. Given a series of adapters $\{\mathcal{A}^i\}$ of length K, the output y for a given input x and parameters W and b is defined as:

$$y = \mathcal{A}^K\left(\mathcal{A}^{K-1}\left(\cdots \mathcal{A}^1(W)\right)\right)x + \mathcal{A}^K\left(\mathcal{A}^{K-1}\left(\cdots \mathcal{A}^1(b)\right)\right)$$

However, we note that some operations within adapters may be non-differentiable, such as binning in quantization. Therefore, we introduce staging, which trains subsequences of adapters in the chain in sequential stages. Intuitively, the chain is broken down into stages according to the regular expression $(\mathcal{A}_n^*\mathcal{A}_d^*)$, where $\mathcal{A}_n$ is the set of non-differentiable adapters and $\mathcal{A}_d$ is the set of differentiable adapters. In this process, each stage is trained separately, where the output of the previous stage is the input to the next. We formalize this process in Appendix A.10.

3.3. Learning Rates

We observe (see Table 4 in Appendix A.8) that different PEFT techniques require significantly different learning rates to achieve the best performance. The optimal learning rates differ by order of magnitude, ranging from 0.0004 (LoRA) to 0.1 (VeRA). Therefore, using a universal learning rate for all PEFT techniques is sub-optimal (see ablation study in Appendix A.9). Instead, in CLAM, we propose a global learning rate scale c as the single learning-rate-related hyperparameter. For each adapter module i, we use $lr_i = c \cdot lr_i^{optimal}$ as the learning rate for its weights, where $lr_i^{optimal}$ is an empirically obtained optimal learning rate based on experiments when adapter i is applied individually.

3.4. Implementation

We implement CLAM in Python with direct integration with Hugging Face's `transformers` library. We use PyTorch to interface with `transformers`, implement techniques, and handle differentiation with multiple learning rates; all reformulated PEFT and pruning techniques are implemented this way. Since quantization works most efficiently when written in CUDA, we rely on the `bitsandbytes` library for sub-implementations of `NF4`, `FP4`, and `FP8`. For a detailed overview of the programmatic abstractions, refer to Appendix A.11. CLAM's recursive mechanics allow for a clean and modular interface for combining adapter techniques. Consequently, adding new techniques does not require considering the compatibility of existing techniques, as this is automatically handled by the code. We plan to open-source this library along with all techniques and expand upon it with the addition of any new method that can be reformulated in our abstraction.

Table 2. **Results for combining PEFT techniques (upper) and compression techniques (lower)**.

		Method	Bit reduction	RTE	MRPC	STS-B	CoLA	CB	Average
PEFT-ONLY	T5-BASE	$(IA)^3$	—	$67.7_{6.8}$	$88.5_{1.5}$	$86.9_{0.5}$	$54.8_{1.4}$	$92.9_{1.8}$	78.2
		LoRA	—	$74.3_{1.3}$	$91.2_{0.7}$	$88.4_{0.4}$	$55.5_{1.0}$	$93.4_{2.7}$	80.56
		LoHA	—	$70.9_{5.1}$	$89.4_{0.6}$	$89.1_{0.5}$	$57.5_{0.8}$	$93.4_{1.0}$	80.0
		VeRA	—	$71.2_{1.2}$	$89.5_{0.7}$	$87.0_{0.1}$	$55.0_{0.5}$	$90.5_{2.1}$	78.6
		CLAM-2	—	$69.3_{3.0}$	$91.0_{0.9}$	$88.4_{0.5}$	$56.5_{1.4}$	$88.1_{3.7}$	78.7
		CLAM-3	—	$\mathbf{77.0_{2.0}}$	$\mathbf{91.5_{0.7}}$	$\mathbf{89.3_{0.5}}$	$\mathbf{60.2_{0.8}}$	$\mathbf{94.0_{2.1}}$	**81.6**
COMPRESSION	GEMMA-2B	$Q(IA)^3$	72.9	$82.4_{0.6}$	$85.0_{1.5}$	$89.3_{0.4}$	$60.3_{1.3}$	$70.8_{2.7}$	77.6
		QLoRA	72.6	$82.7_{1.0}$	$85.6_{1.2}$	$89.8_{0.8}$	$65.5_{1.0}$	$73.2_{1.8}$	79.4
		QLoHA	72.2	$58.5_{9.7}$	$82.4_{1.9}$	$89.8_{0.3}$	$62.4_{1.1}$	$67.3_{5.5}$	72.1
		QVeRA	72.9	$78.9_{2.8}$	$86.3_{3.7}$	$88.0_{2.0}$	$59.5_{2.4}$	$70.2_{4.1}$	76.58
		PLoRA	49.6	$54.3_{1.8}$	$85.1_{0.9}$	$90.9_{0.0}$	$\mathbf{66.1_{0.8}}$	$67.3_{4.5}$	72.7
		CLAM-Q›3	72.2	$\mathbf{84.4_{1.7}}$	$\mathbf{86.9_{0.5}}$	$\mathbf{90.9_{0.1}}$	$64.3_{1.6}$	$74.4_{3.7}$	**80.2**
		CLAM-PQ›3	**85.7**	$83.9_{2.9}$	$84.2_{1.2}$	$89.6_{1.1}$	$59.2_{2.4}$	$\mathbf{79.9_{9.3}}$	79.4

4. Experiments

We seek to answer the following research questions. **RQ1:** Can CLAM produce better PEFT compositions than individual techniques? **RQ2:** Can CLAM compose quantization, pruning, and PEFT techniques to create new state-of-the-art techniques for compressing and finetuning models? **RQ3:** Are CLAM chains more parameter efficient?

4.1. Experimental Setup

Models, compute, and datasets. We evaluate CLAM on two models with different architecture and size. First, we apply CLAM to T5-BASE (Raffel et al., 2020), an encoder-decoder LLM with 220M parameters. Thereafter, we move to GEMMA-2B (Team et al., 2024), a decoder-only model. Both models are fitted with classification heads to interface with the benchmarks. All experiments were run on a machine with 8 NVIDIA A100 GPUs. Learning rates were set on a per-experiment basis. We train these models on the GLUE (Wang et al., 2018) benchmarks CoLA, MRPC, RTE, and STS-B and the SuperGLUE (Wang et al., 2019) benchmark CB.

Evaluation. For our configuration searches, we generate a train and validation split of the provided training sets. The provided test sets are held out for validation after optimal configurations are determined. Details on these splits can be found in Appendix A.8. We report Pearson correlation for STS-B, Matthew's correlation for CoLA, and evaluation accuracy for the rest of the benchmarks.

Search and compression. We employ the Tree-structured Parzen Estimator (Bergstra et al., 2011) algorithm via the Optuna framework to explore the search space consisting of technique combinations, learning rate scales, and individual ranks. The value that is optimized is the evaluation performance on the training holdout set. Each trial consists of applying a single CLAM stage followed by some finetuning epochs (the details of which can be found in Appendix A.8), where the training holdout is evaluated after each epoch. To avoid returning overfit results to the search algorithm, we report the best holdout score over all epochs. We run CLAM for at most 120 trials, which is 12% or 0.7% of the total search space for chains of length 2 and 3 respectively. We use NF4 (Dettmers et al., 2024) for quantization and WANDA (Sun et al., 2023) for pruning.

4.2. RQ1: Discovering Better PEFT Methods

Baselines. We use individual PEFT techniques $(IA)^3$, LoRA, LoHa, and VeRA as finetuning baselines, and evaluate their performance when applied on T5-BASE. We perform grid search on learning rates and ranks individually for each technique over all datasets. Results of baselines are presented in Table 2. The respective optimal learning rates are presented in Table 4 of Appendix A.8, which will be used as the base learning rate in CLAM.

Setup. We show the potential of chaining PEFT techniques by presenting results from the best combinations of lengths 2 and 3, denoted by CLAM-2 and CLAM-3. For CLAM trials, we search for the global learning rate scale c from a discrete set $\{0.1, 0.4, 1, 4\}$, set $lr_i^{optimal}$ for each individual technique i as shown in Table 4, and search the individual rank from a discrete set $\{4, 8, 16, 32, 64\}$. For CLAM-2, the search space is of size 1024, and CLAM-3 it is 16384. The best configuration is selected based on the validation results by searching for up to 120 trials using Optuna, and results evaluated on test sets are reported in Table 2.

Results. CLAM-3 performs consistently better than the best of individual methods over all datasets, while only exploring 0.73% (120/16384) of the search space. In comparison, we perform full grid search for all hyperparameters of individual methods. Notably, we limit the search space for the rank used by CLAM so that CLAM-2 and CLAM-3 would not use more trainable parameters than the highest budget provided to baselines. Under the same (or less) budget, and with less hyperparameter search, CLAM-3's superiority over baselines demonstrates the efficacy of the design space enabled by CLAM. Interestingly, CLAM-2 only performs better than baselines on one out of five datasets. We attribute this result to the limited design space of CLAM-2. In fact, the performance gap between CLAM-2 and CLAM-3 conversely shows that a higher-order design space is desirable.

Table 3. **Control experiments with same parameter / bit budget.**

Method	RTE	MRPC	STS-B	CoLA	CB	Average
PARAMS (T5-BASE)						
LoRA	$61.0_{4.9}$	$91.2_{0.7}$	$87.7_{1.8}$	$55.5_{1.0}$	$93.4_{2.7}$	77.7
LoHA	$70.9_{5.2}$	$79.5_{0.7}$	$89.1_{0.8}$	$57.5_{0.9}$	$93.4_{1.0}$	78.1
CLAM-3	$\mathbf{73.4_{0.9}}$	$\mathbf{91.5_{0.7}}$	$\mathbf{89.3_{0.3}}$	$\mathbf{60.1_{0.8}}$	$\mathbf{94.0_{2.1}}$	**81.7**
BITS (GEMMA 2B)						
CLAM-P(86%)-3	$54.2_{1.7}$	$9.1_{2.9}$	$68.1_{1.4}$	$41.8_{20.7}$	$73.8_{1.0}$	49.4
CLAM-PQ›3	$\mathbf{83.9_{2.9}}$	$\mathbf{84.2_{1.2}}$	$\mathbf{89.6_{1.1}}$	$\mathbf{59.2_{2.4}}$	$\mathbf{79.9_{9.3}}$	**79.4**

4.3. RQ2: Chaining for Better Compression

We start with combining quantization with PEFT techniques to seek the optimal quantization method. Then we further introduce pruning to pursue the optimal compression method.

Baselines. For the optimal quantization experiment, we observe how CLAM combines PEFT with quantization; therefore, we compare with the SOTA technique QLoRA (Dettmers et al., 2024) as implemented in CLAM, as well as its variants QLoHA, QVeRA, and Q(IA)^3 that are naturally enabled by our framework. For the optimal compression experiment, we observe how CLAM performs in combining quantization, pruning, and PEFT. As such, we compare CLAM to QLoRA and Wanda-LoRA (denoted PLoRA).

Setup. For the optimal quantization experiment, we use CLAM to search PEFT chains of length 3; however, unlike the optimal PEFT experiment, we prepend a quantization adapter to the chain, denoted as CLAM-Q›3. This gives us a search space of size 16384, of which we search 120 trials per benchmark. For the optimal compression experiment, we take the best chain per benchmark from the optimal quantization experiment's results and add a pruning adapter to the beginning of the chain, denoted as CLAM-PQ›3, effectively combining quantization, pruning, and PEFT.

Results for optimal quantization. Table 2 presents results for the baselines and CLAM. The optimal quantization shows that CLAM created chains surpassed the baselines on performance on four of five benchmarks, including QLoRA. This result is despite only searching 0.73% percent of the chain search space. It provides evidence that CLAM's chaining method can be more effective than any standalone technique for finetuning models that have been quantized.

Results for optimal compression. Providing further insight into CLAM's ability to compress and finetune models, the optimal compression experiment (Table 2) introduces pruning. It shows that, despite its backbone having 50.0% fewer non-zero bits than CLAM-Q›3, it performs to within 1% on average. Furthermore, it averaged roughly the same score as QLoRA and performed 6.7% better than PLoRA despite having 50.0% and 73.1% fewer non-zero bits, respectively. This may be counterintuitive, as one may expect a combined technique to perform roughly on par with the lesser of the pair; however, we contend that this CLAM chain could have subtle benefits, which are twofold. First, pruning and quantization can have a combined regularizing effect on the model. Second, and more importantly, differentiable CLAM PEFT chains are inherently more mathematically expressive than LoRA and thus can better make up for the information loss introduced by compression.

4.4. RQ3: Parameter Efficiency of CLAM Chains

PEFT techniques. We conduct control experiments to show that chained techniques perform significantly better than their individual component techniques under the same trainable parameter count. This experiment is conducted on a per-benchmark basis. Specifically, we set LoRA and LoHa's rank accordingly so that LoRA and LoHa use the same number of trainable parameters as CLAM-3 for that specific chain on that benchmark (as shown in Table 8). VeRA is not included here since the parameter count is far lower than other techniques and to provide the same budget would require an impractical r value in the tens of thousands. As presented in the upper half table of Table 3, with the same parameter count, CLAM-3 significantly outperforms individual techniques, showing that CLAM-3 can utilize trainable parameters more effectively, and combining multiple techniques has advantages.

Compression. Similarly, we conduct experiments to show that with the same number of active bits (same compression ratio), pruning with quantization performs significantly better than pruning alone. We set the sparsity ratio of WANDA to 86% so that the number of active bits matches CLAM-PQ›3. Quantization alone is not included as a baseline here since there is no hyperparameter to control bit reduction. As presented in the lower half of Table 3, pruning at a high sparsity ratio induces significant performance loss. On the contrary, CLAM-PQ›3 can largely retain the performance. This result clearly shows that the combination of multiple compression techniques is highly beneficial.

5. Conclusion

We presented CLAM, a framework for unlimited chaining of adapters across PEFT, quantization, and pruning. The key idea is that all techniques are reformulated as adapters that perform weight modification according to a shared interface. We demonstrate that CLAM composes new PEFT techniques that outperform their individual components; new quantization-aware finetuning methods that outperform the state-of-the-art; and a new compression method at the intersection of all three fields which performs competitively with pruning or quantization alone, but for significantly less information-holding bits. However, CLAM has design and computational limitations, which we elaborate in Appendix A.7. In the future, we plan to expand our framework with emerging techniques and open-source our library with direct compatibility with modern software stacks.

References

Bergstra, J., Bardenet, R., Bengio, Y., and Kégl, B. Algorithms for hyper-parameter optimization. In Shawe-Taylor, J., Zemel, R., Bartlett, P., Pereira, F., and Weinberger, K. (eds.), *Advances in Neural Information Processing Systems*. Curran Associates, Inc., 2011.

Chen, J., Zhang, A., Shi, X., Li, M., Smola, A., and Yang, D. Parameter-efficient fine-tuning design spaces. In *International Conference on Learning Representations*, 2022.

Dettmers, T., Lewis, M., Belkada, Y., and Zettlemoyer, L. Llm.int8(): 8-bit matrix multiplication for transformers at scale. *Advances in Neural Information Processing Systems*, 2022.

Dettmers, T., Svirschevski, R. A., Egiazarian, V., Kuznedelev, D., Frantar, E., Ashkboos, S., Borzunov, A., Hoefler, T., and Alistarh, D. Spqr: A sparse-quantized representation for near-lossless llm weight compression. In *International Conference on Learning Representations*, 2023.

Dettmers, T., Pagnoni, A., Holtzman, A., and Zettlemoyer, L. Qlora: Efficient finetuning of quantized llms. *Advances in Neural Information Processing Systems*, 2024.

Frantar, E. and Alistarh, D. Qmoe: Practical sub-1-bit compression of trillion-parameter models. *arXiv preprint arXiv:2310.16795*, 2023a.

Frantar, E. and Alistarh, D.-A. Sparsegpt: Massive language models can be accurately pruned in one-shot. In *International Conference on Machine Learning*, 2023b.

Frantar, E., Ashkboos, S., Hoefler, T., and Alistarh, D. Gptq: Accurate post-training quantization for generative pre-trained transformers. *International Conference on Learning Representations*, 2022.

GGML. Ggml: Tensor library for machine learning, 2024. URL `https://github.com/ggerganov/ggml`.

Han, Z., Gao, C., Liu, J., Zhang, S. Q., et al. Parameter-efficient fine-tuning for large models: A comprehensive survey. *arXiv preprint arXiv:2403.14608*, 2024.

He, J., Zhou, C., Ma, X., Berg-Kirkpatrick, T., and Neubig, G. Towards a unified view of parameter-efficient transfer learning. In *International Conference on Learning Representations*, 2022.

Hu, E. J., Wallis, P., Allen-Zhu, Z., Li, Y., Wang, S., Wang, L., Chen, W., et al. Lora: Low-rank adaptation of large language models. In *International Conference on Learning Representations*, 2021.

Hyeon-Woo, N., Ye-Bin, M., and Oh, T.-H. Fedpara: Low-rank hadamard product for communication-efficient federated learning. *arXiv preprint arXiv:2108.06098*, 2021.

Kalajdzievski, D. A rank stabilization scaling factor for fine-tuning with lora. *arXiv preprint arXiv:2312.03732*, 2023.

Kopiczko, D. J., Blankevoort, T., and Asano, Y. M. VeRA: Vector-based random matrix adaptation. In *The Twelfth International Conference on Learning Representations*, 2024.

Kwon, W., Kim, S., Mahoney, M. W., Hassoun, J., Keutzer, K., and Gholami, A. A fast post-training pruning framework for transformers. *Advances in Neural Information Processing Systems*, 2022.

Lester, B., Al-Rfou, R., and Constant, N. The power of scale for parameter-efficient prompt tuning. In *Empirical Methods in Natural Language Processing*, 2021.

Li, Y., Yu, Y., Liang, C., Karampatziakis, N., He, P., Chen, W., and Zhao, T. Loftq: Lora-fine-tuning-aware quantization for large language models. In *The Twelfth International Conference on Learning Representations*, 2023.

Lin, J., Tang, J., Tang, H., Yang, S., Dang, X., and Han, S. Awq: Activation-aware weight quantization for llm compression and acceleration. *arXiv preprint arXiv:2306.00978*, 2023.

Liu, H., Tam, D., Muqeeth, M., Mohta, J., Huang, T., Bansal, M., and Raffel, C. A. Few-shot parameter-efficient fine-tuning is better and cheaper than in-context learning. *Advances in Neural Information Processing Systems*, 2022.

Liu, S.-Y., Wang, C.-Y., Yin, H., Molchanov, P., Wang, Y.-C. F., Cheng, K.-T., and Chen, M.-H. Dora: Weight-decomposed low-rank adaptation. *arXiv preprint arXiv:2402.09353*, 2024.

Ma, X., Fang, G., and Wang, X. Llm-pruner: On the structural pruning of large language models. *Advances in neural information processing systems*, 2023.

Mao, Y., Mathias, L., Hou, R., Almahairi, A., Ma, H., Han, J., Yih, S., and Khabsa, M. Unipelt: A unified framework for parameter-efficient language model tuning. In *Proceedings of the Association for Computational Linguistics*, 2022.

Raffel, C., Shazeer, N., Roberts, A., Lee, K., Narang, S., Matena, M., Zhou, Y., Li, W., and Liu, P. J. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of Machine Learning Research*, 2020.

Sun, M., Liu, Z., Bair, A., and Kolter, J. Z. A simple and effective pruning approach for large language models. In *International Conference on Learning Representations*, 2023.

Team, G., Mesnard, T., Hardin, C., Dadashi, R., Bhupatiraju, S., Pathak, S., Sifre, L., Rivière, M., Kale, M. S., Love, J., et al. Gemma: Open models based on gemini research and technology. *arXiv preprint arXiv:2403.08295*, 2024.

Valipour, M., Rezagholizadeh, M., Kobyzev, I., and Ghodsi, A. Dylora: Parameter efficient tuning of pre-trained models using dynamic search-free low-rank adaptation. *arXiv preprint arXiv:2210.07558*, 2022.

Wang, A., Singh, A., Michael, J., Hill, F., Levy, O., and Bowman, S. GLUE: A multi-task benchmark and analysis platform for natural language understanding. In Linzen, T., Chrupała, G., and Alishahi, A. (eds.), *Proceedings of the 2018 EMNLP Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*, November 2018.

Wang, A., Pruksachatkun, Y., Nangia, N., Singh, A., Michael, J., Hill, F., Levy, O., and Bowman, S. Superglue: A stickier benchmark for general-purpose language understanding systems. *Advances in Neural Information Processing Systems*, 32, 2019.

Yang, A. X., Robeyns, M., Wang, X., and Aitchison, L. Bayesian low-rank adaptation for large language models. In *International Conference on Learning Representations*, 2023.

Zhang, M., Shen, C., Yang, Z., Ou, L., Yu, X., Zhuang, B., et al. Loraprune: Pruning meets low-rank parameter-efficient fine-tuning. *arXiv preprint arXiv:2305.18403*, 2023a.

Zhang, Y., Zhao, L., Lin, M., Yunyun, S., Yao, Y., Han, X., Tanner, J., Liu, S., and Ji, R. Dynamic sparse no training: Training-free fine-tuning for sparse llms. In *International Conference on Learning Representations*, 2023b.

Zhou, H., Wan, X., Vulić, I., and Korhonen, A. Autopeft: Automatic configuration search for parameter-efficient fine-tuning. *Transactions of the Association for Computational Linguistics*, 2024.

A. Appendix

A.1. Quantization

Quantization converts values from a high-precision data type to a lower-precision data type. For transformers, this is typically a lossy mapping of a tensor's M-bit floating-point element $X^{\text{FP}(M)} \in \mathbb{R}$ to an N-bit integer $X^{\text{Int}(N)} \in \{0 \dots 2^N - 1\}$, where $M \in \{16, 32, 64\}$ and $N \in \{1, 2, 4, 8\}$. This requires binning of the form:

$$X^{\text{Int}(N)} = Q(X^{\text{FP}(M)}) = \text{round}\left((2^N - 1)F\left(X^{\text{FP}(M)}\right)\right)$$

where $F : \mathbb{R} \to [0, 1]$ is a cumulative distribution function. Determining the optimal F at varying bits N is a significantly active area of research. For example, QLoRA (Dettmers et al., 2024) proposes a normal F that assigns floats to 4-bit NormalFloat (NF4) bins.

As explained in Appendix A.5, quantization can break structural invariants of data types and "un-prune" zeroed-out weights. Quantization frameworks thus provide dequantization functions so that the data type conversion is approximately revertible:

$$X^{\text{FP}(M)} \approx Q^{-1}(X^{\text{Int}(N)}) = F^{-1}\left(X^{\text{Int}(N)}/(2^N - 1)\right)$$

A.2. Pruning

Pruning aims to speed up matrix storage and multiplication with sparsification by setting a certain percentage of the matrix elements to zero. A general pruning process on a linear layer can be formulated as:

$$y = (M \odot W)x + b$$

where M is a Boolean mask of the same shape as W. Several methods aim to prune as many parameters with as little performance degradation as possible (Frantar & Alistarh, 2023b; Ma et al., 2023; Sun et al., 2023). Pruning is typically done in a structure (i.e. in specific GPU-advantaged patterns), which introduces complications for subsequent techniques, such as quantization or PEFT, which should not disrupt this pattern; in reverse order, to combine properly, a pruning technique should be able to operate on quantized data structures.

A.3. PEFT Techniques

(IA)^3. (IA)^3 (Liu et al., 2022) is a PEFT method that learns rescaling factors for activations in the network. Given a frozen weight $W_i \in \mathbb{R}^{m \times n}$, input $x \in \mathbb{R}^{n \times 1}$, and rescaling factor $l_i \in \mathbb{R}^{m \times 1}$, it will directly rescale the output of a projection: $y = (W_i x) \odot l_i$. While straightforward, combining it with another PEFT method is complicated because it is multiplying the hidden state, rather than adding to it. Say we want to combine this technique with another that adds to the activation; it is ambiguous whether this additive activation should be multiplied by IA^3, in addition to the original activation, or not. Moreover, pruning and quantization may compromise the adapter, since it is especially information-dense. To resolve this, one may choose to merge the adapter into the weight, but if it is being combined with another PEFT technique, merging may not be possible due to dimension or type mismatch.

Low-rank adaptation (LoRA). LoRA (Hu et al., 2021) inserts two trainable low-rank matrices $A_i \in \mathbb{R}^{m \times r}$ and $B_i \in \mathbb{R}^{r \times n}$ for every targeted weight $W_i \in \mathbb{R}^{m \times n}$ in a pretrained model. During a forward pass of the model, they are used to construct an additional projection that is added to the hidden state in the form: $y = W_i x + cA_i B_i x$, where $c \in \mathbb{R}$ is a constant. During finetuning, only the parameters A_i and B_i are updated. LoRA can be problematic in combining with other techniques because it introduces entirely new weight matrices that must be accounted for by other adapters (see (IA)^3 for instance). It is an important design decision whether or not these new matrices should be quantized or pruned, as doing so could impact performance significantly given their higher-than-average information density. As mentioned above, it may not be possible to merge LoRA when it is combined with other techniques if those other techniques are not mergeable; even if merging was possible, the merging may be lossy as LoRA weights are typically stored in higher precision than model weights.

Low-rank Hadamard product (LoHA). Proposed by Hyeon-Woo et al. (2021), LoHA is similar to LoRA with one exception: they introduce additional matrices that are combined via a Hadamard product. More specifically, given a target weight W_i, LoHA introduces the low-rank matrices $A_{i,1}, A_{i,2} \in \mathbb{R}^{m \times r}$ and $B_{i,1}, B_{i,2} \in \mathbb{R}^{r \times n}$ in: $y = W_i x + c(A_{i,1}B_{i,1} \odot$

$A_{i,2}B_{i,2})x$. This Hadamard product is intended to provide additional expressivity. This PEFT technique holds similar challenges for being combined as with LoRA.

Vector-based random matrix adaptation (VeRA). VeRA (Kopiczko et al., 2024) bears strong similarities to both LoRA and (IA)^3. It randomly generates a pair of low-rank matrices shared across all VeRA adapters, each individually learning an associated rescaling of these random matrices. Given a frozen weight W_i, shared frozen matrices $A \in \mathbb{R}^{m \times r}$ and $B \in \mathbb{R}^{r \times n}$, and trainable diagonal matrices $\Lambda_{A,i}$ and $\Lambda_{B,i}$, VeRA add to the original projection via: $y = (W_i x) + \Lambda_{A,i} A \Lambda_{B,i} B$ This method contains the challenges of both LoRA and (IA)^3 simultaneously. The scaling diagonals are sensitive to both pruning as well as quantization, and the added operations within this method make combining with other PEFT techniques more ambiguous and challenging.

A.4. Standalone Pairs

Recent works have successfully demonstrated that combining techniques is effective in minimizing the number of bits in a model while maximizing parameter-efficient finetuning performance. One method, QLoRA (Dettmers et al., 2024), combines NF4 quantization with LoRA; given a weight matrix W_i that has been quantized to D_i according to NF4 quantization, QLoRA computes the following on a forward pass:

$$y = Q_{NF4}^{-1}(D_i)x + cA_iB_ix$$

where Q_{NF4}^{-1} is the NF4 dequantize function. A similar method, Wanda-LoRA (Sun et al., 2023), prunes the matrix W_i to get P_i in:

$$y = P_i x + cA_iB_ix.$$

A.5. Challenges in Combining Techniques

Quantization-first. Popular quantization techniques such as NF4 (Dettmers et al., 2024) do not preserve the structural invariants of the data being quantized. In NF4's current implementation, quantized matrices of shape (m, n) are stored as Int8 vectors of length $mn/2$, where each element represents two NF4 values. Because the tensor shape is changed, NF4 does not preserve a one-to-one mapping between elements in the original and quantized tensors. This is problematic for PEFT adapters and pruners that seek to directly manipulate possibly-quantized weights since they must account for all possibilities of data types and structures as well as their operator semantics. Thus, in a quantization-first combination, subsequent adapters must accommodate the changes in the pretrained matrix's shape, data type, and operational semantics.

Quantization–Pruning. When combining quantization and pruning, the pruning method must operate over the structural changes induced by quantization as previously mentioned. On the other hand, quantization must be able to reconstruct zero-valued elements perfectly in order to preserve the pruned weight structure.

PEFT–Pruning. When combining PEFT and pruning, any PEFT adapter that merges back into the pretrained model must not change sparsified matrix entries. Lastly, consider combining techniques within PEFT; some techniques operate by multiplying the hidden state (e.g. (IA)^3), while others add to it; in this case, the order of operations when combining two techniques matters.

Algorithmic ambiguities. Foremost, there is the choice of invariants a technique must satisfy (or modify to satisfy) to be chainable. Further algorithmic ambiguities are revealed by considering the example pipeline in Figure 1. VeRA (`d2`) is followed by pruning (`n4`), but should the reparameterized VERA weights be pruned before, during, or after fine-tuning? Note also that the PEFT method LoRA (`d1`) is not applied in isolation. Learnable scaling vectors are subsequently applied by VeRA and IA^3 (`d3`), and it is not obvious whether they should rescale the LoRA-reparameterized weights, or the merged weights after LoRA fine-tuning. More generally, it needs to be specified whether "applying" a PEFT method constitutes just injecting adapter weights into the model, or also includes the fine-tuning step.

A.6. Related Work

Unified PEFT frameworks. He et al. (2022) and Mao et al. (2022) propose unified perspectives of PEFT by implementing hybrid architectures for tuning custom ensembles of PEFT modules. Chen et al. (2022) and Zhou et al. (2024) present methods of searching for new PEFT designs over a unified space of configurations. CLAM expands on these frameworks by enabling the unlimited nesting of PEFT modules within each other. Our work further broadens the space of unifiable techniques by supporting quantization and pruning techniques in addition to PEFT.

Combining LLM techniques. Combinatory solutions from Dettmers et al. (2024) and Li et al. (2023) exploit both the fine-tuning efficiency of LoRA and the memory savings from quantization. Similar benefits from joining LoRA with pruning are explored by Sun et al. (2023) and Zhang et al. (2023a). CLAM subsumes many of these combinations and facilitates the discovery of future ones by introducing freely chainable techniques drawn from an extensible library of adapters.

Quantization. Works such as GPTQ (Frantar et al., 2022), SpQR (Dettmers et al., 2023), GGML (GGML, 2024), and AWQ (Lin et al., 2023) quantize to around 4 bits, in some instances going as low as 2 bits. Other techniques, such as QMoE (Frantar & Alistarh, 2023a), focus on a specific architecture and are able to achieve sub 1-bit quantization. CLAM is designed to be agnostic to any particular quantization method, and it is future work to subsume all of these.

Pruning. The main bottleneck to adopting neural pruning techniques for billion-parameter LLMs is the computationally intensive retraining step for recovering performance after pruning. Thus, there is significant research interest in retraining-free pruning methods, such as WANDA (Sun et al., 2023), SparseGPT (Frantar & Alistarh, 2023b), and the frameworks of Kwon et al. (2022) and Zhang et al. (2023b). Other approaches employ a parameter-efficient method like LoRA for retraining (Ma et al., 2023; Zhang et al., 2023a). CLAM's interface is designed to integrate both of these directions.

PEFT. Two major designs are additive PEFT and reparameterized PEFT (Han et al., 2024). Additive PEFT methods such as soft-prompting (Lester et al., 2021) and $(IA)^3$ (Liu et al., 2022) inject new parameters (e.g. learnable scaling vectors) into transformer modules. Reparameterized PEFT decomposes weight matrices into lower-dimensional forms for tuning; this is explored by LoRA (Hu et al., 2021) and its derivatives (Valipour et al., 2022; Yang et al., 2023; Kalajdzievski, 2023), as well as LoHA (Hyeon-Woo et al., 2021), VeRA (Kopiczko et al., 2024) and DoRA (Liu et al., 2024). CLAM unifies additive and reparameterized techniques in an interoperable manner.

A.7. Limitations

While CLAM was designed to interface with as many emerging PEFT techniques as possible, by design, it cannot limitlessly chain those that are not mergeable (i.e., there is no computable W' emulating its modifications to some given tensor W). This might leave even simple techniques, such as those using non-linear functions, incompatible. Integrating these modules into CLAM chains is necessary for maximum subsumption and likewise an intended future direction.

Not every chain is a valid one. For example, quantizing in a chain more than once is undefined behavior, requiring the quantization algorithm to be able to operate over the data structure it was designed to produce. Also, there is no hard requirement that every quantization algorithm preserve zeros for pruning. Therefore, there needs to be a more nuanced accommodation for technique-specific restrictions.

CLAM focuses on the linear layer for maximum compatibility across models. However, doing so neglects other large transformers, such as emerging multi-modal models, that employ convolutional layers. It is worth studying how CLAM chains could have an impact on these other types of models and their respective fields by supporting these other layer types.

Finally, §4.2 demonstrates that CLAM chains of length 3 can have tens of thousands of possible configurations. This space is intractable to search with our current methods. Therefore, principled explorations of this newly introduced design space are a new opportunity for future research.

A.8. Experimental Setup

Table 4. Optimal learning rates by methods.

Method	**RQ1**	**RQ2**	**RQ3 (T5-base)**	**RQ3 (Gemma 2B)**
$(IA)^3$	1e-2	1e-2	1e-2	1e-2
LoRA	4e-4	4e-4	4e-4	4e-4
LoHA	4e-3	1e-3	4e-3	1e-3
VeRA	1e-1	1e-1	1e-1	1e-1

Table 5. Training Settings for Baselines

Training setting	**Configuration**
optimizer	`paged_adamw_32bit`
weight decay	0.06
optimizer momentum	β_1, β_2 = 0.9, 0.999
batch size	32
training epochs	10
learning rate schedule	linear
warmup steps	0
max gradient norm	1.0

Table 6. The dataset sizes for the holdout set for various benchmarks. `search-train` and `search-val` are the splits we applied on the original training sets from GLUE and SuperGLUE for the purposes of hyperparameter and configuration searching. `eval-train` is the entire training set and `eval-test` is the holdout test set for the purposes of validation.

Dataset	**#search-train**	**#search-val**	**#eval-train**	**#eval-test**
RTE	1,992	498	2,490	277
MRPC	2,934	734	3,668	408
STS-B	4,599	1,150	5,749	1,500
CoLA	6,841	1,710	8,551	1,043
CB	200	50	250	56

Table 7. Baseline configurations used across various tasks, found via hyperparameter search. Note that quantized methods (denoted as beginning with 'Q'), values were used for both the optimal quantization and compression experiments.

Task		**$(IA)^3$**	**LoRA**	**LoHA**	**VeRA**	**$Q(IA)^3$**	**QLoRA**	**QLoHA**	**QVeRA**	**PLoRA**
RTE	**Rank**	—	64	8	64	—	8	64	64	8
	LR	4e-2	4e-4	1e-2	1e-1	1e-2	1e-3	4e-3	1e-4	1e-3
	Train params	0.13M	3.54M	1.77M	0.06M	0.31M	0.93M	14.75M	0.05M	0.93M
MRPC	**Rank**	—	16	16	32	—	64	32	64	64
	LR	1e-2	4e-4	4e-3	1e-1	1e-2	1e-4	4e-3	4e-2	1e-4
	Train params	0.13M	1.77M	3.54M	0.06M	0.31M	7.38M	7.38M	0.05M	7.38M
STS-B	**Rank**	—	8	32	32	—	32	16	32	32
	LR	4e-2	1e-3	4e-3	4e-1	1e-2	4e-4	4e-3	4e-2	4e-4
	Train params	0.13M	0.88M	7.08M	0.06M	0.31M	3.69M	3.69M	0.04M	3.69M
CoLA	**Rank**	—	32	16	64	—	64	8	8	64
	LR	1e-2	4e-4	4e-3	1e-1	1e-2	1e-4	4e-3	4e-2	1e-4
	Train params	0.13M	3.54M	3.54M	0.06M	0.31M	7.38M	1.85M	0.05M	7.38M
CB	**Rank**	—	4	32	64	—	64	8	8	64
	LR	4e-2	1e-2	4e-2	1e-1	1e-3	1e-3	1e-3	1e-1	1e-3
	Train params	0.13M	0.44M	7.08M	0.06M	0.31M	7.38M	1.85M	0.05M	7.38M

Table 8. CLAM configurations used across various tasks, found via search over the configuration space. The quantization method used is NF4, and the pruning method used is WANDA. 'Q' denotes that quantization was applied before the PEFT chain. 'PQ' denotes that pruning was applied first and then quantization second before the PEFT chain. Note that 'PQ' re-uses configurations found for 'Q'. All PEFT methods in CLAM-Q and CLAM-PQ chains use hyperparameters for their quantized forms found in Table 7.

Task	**Method**	**PEFT 1**	**PEFT 2**	**PEFT 3**	**LR scale**	**Train params**
RTE	CLAM-2	LoRA (64)	VeRA (4)	—	1.0	7.13M
	CLAM-3	$(IA)^3$	$(IA)^3$	$(IA)^3$	1.0	14.34M
	CLAM-Q›3	LoRA (16)	LoRA (4)	$(IA)^3$	1.0	2.61M
	CLAM-PQ›3	LoRA (16)	LoRA (4)	$(IA)^3$	1.0	2.61M
MRPC	CLAM-2	$(IA)^3$	LoHA (16)	—	1.0	3.67M
	CLAM-3	$(IA)^3$	LoRA (32)	$(IA)^3$	1.0	3.80M
	CLAM-Q›3	VeRA (4)	LoRA (64)	VeRA (64)	0.4	7.46M
	CLAM-PQ›3	VeRA (4)	LoRA (64)	VeRA (64)	0.4	7.46M
STS-B	CLAM-2	LoRA (32)	LoHA (4)	—	0.4	4.42M
	CLAM-3	VeRA (32)	$(IA)^3$	LoRA (64)	1.0	7.26M
	CLAM-Q›3	VeRA (64)	LoHA (16)	LoRA (16)	1.0	5.57M
	CLAM-PQ›3	VeRA (64)	LoHA (16)	LoRA (16)	1.0	5.57M
CoLA	CLAM-2	LoRA (32)	$(IA)^3$	—	1.0	3.67M
	CLAM-3	$(IA)^3$	LoRA (32)	$(IA)^3$	0.4	3.80M
	CLAM-Q›3	VeRA (64)	LoHA (16)	LoRA (16)	1.0	5.84M
	CLAM-PQ›3	VeRA (64)	LoHA (16)	LoRA (16)	1.0	5.84M
CB	CLAM-2	VeRA (64)	LoRA (64)	—	1.0	6.19M
	CLAM-3	$(IA)^3$	LoHA (4)	VeRA (4)	4.0	1.07M
	CLAM-Q›3	LoRA (64)	LoRA (64)	LoRA (4)	1.0	15.21M
	CLAM-PQ›3	LoRA (64)	LoRA (64)	LoRA (4)	1.0	15.21M

A.9. Ablation Study

Fixed PEFT hyperparameters. In our first ablation, we simplified our search over the different orderings of PEFT techniques by fixing the decomposition rank and learning rates of each technique to be the optimal ones found from our baseline search in Table 7. For example, LoRA on RTE uses a fixed rank of 64 and learning rate of 4e-4. We call the length 2 and 3 configurations found by this modified search CLAM-2* and CLAM-3*. The decreased performance of CLAM-2* and CLAM-3* compared to CLAM-2 and CLAM-3 on the benchmarks suggests that while the fixed parameters were optimized for individual tasks, they may not be universally effective across different configurations or compositional chains. This suggests the necessity for dynamic parameter adjustment and an extensive search to underscore the rich interactions between different PEFT techniques.

Table 9. Results for fixing hyperparameters during PEFT configuration search. Baseline results are from Table 2.

Method	RTE	MRPC	STS-B	CoLA	CB	Average
$(IA)^3$	$67.7_{6.8}$	$88.5_{1.5}$	$86.9_{0.5}$	$54.8_{1.4}$	$\mathbf{92.9_{1.8}}$	78.2
LoRA	$\mathbf{74.3_{1.3}}$	$89.5_{0.4}$	$\underline{88.4_{0.4}}$	$55.1_{2.0}$	$87.5_{5.4}$	**79.0**
LoHA	$\underline{69.2_{2.3}}$	$89.4_{0.6}$	$\mathbf{89.1_{0.5}}$	$\mathbf{57.3_{0.7}}$	$82.7_{5.7}$	77.5
VeRA	$66.2_{1.5}$	$87.6_{1.3}$	$84.8_{0.9}$	$49.2_{1.1}$	$87.5_{3.1}$	75.1
CLAM-2*	$62.7_{4.5}$	$\mathbf{91.6_{0.5}}$	$86.7_{0.7}$	$56.5_{1.4}$	$84.5_{2.7}$	76.4
CLAM-3*	$68.2_{0.7}$	$\underline{91.2_{0.9}}$	$88.0_{0.9}$	$\underline{56.7_{1.3}}$	$\underline{88.7_{2.1}}$	<u>78.6</u>

Table 10. CLAM configurations found by the PEFT ordering search with hyperparameters fixed.

Task	Method	PEFT 1	PEFT 2	PEFT 3	Train params
RTE	CLAM-2*	LoHA	$(IA3)^3$	—	1.90M
	CLAM-3*	LoHA	LoHA	LoRA	1.06M
MRPC	CLAM-2*	LoHA	LoRA	—	5.31M
	CLAM-3*	$(IA)^3$	LoHA	LoHA	7.21M
STS-B	CLAM-2*	$(IA)^3$	LoRA	—	1.01M
	CLAM-3*	LoRA	$(IA)^3$	LoRA	8.85M
CoLA	CLAM-2*	LoRA	$(IA)^3$	—	3.67M
	CLAM-3*	$(IA)^3$	LoHA	LoHA	7.21M
CB	CLAM-2*	LoHA	LoHA	—	14.16M
	CLAM-3*	VeRA	VeRA	LoHA	7.20M

Global learning rate. Our objective in this ablation was to investigate the effects of applying a single global learning across all PEFT adapters for a given task, as opposed to individual learning rates per PEFT parameter group. We retain an individual search for each technique's optimal rank in this experiment. The global learning rate is searched over the same space of learning rates that we provide to each individual optimization during the Optuna search. We call the length 2 and 3 configurations found by this modified search CLAM-2† and CLAM-3†. We find that CLAM-3† performs 2.2% worse than CLAM-3 while CLAM-2† performs 0.3% better than CLAM-2. This suggests the use of a global learning rate has less effect when there are only 2 optimizations, but with 3 optimizations, each parameter's groups sensitivity to learning rate leads to sub-optimal results.

Table 11. Results for using a global learning rate during PEFT configuration search. Baseline results are from Table 2.

Method	RTE	MRPC	STS-B	CoLA	CB	Average
(IA)^3	$67.7_{6.8}$	$88.5_{1.5}$	$86.9_{0.5}$	$54.8_{1.4}$	$\underline{92.9_{1.8}}$	78.2
LoRA	$\mathbf{74.3_{1.3}}$	$89.5_{0.4}$	$88.4_{0.4}$	$55.1_{2.0}$	$87.5_{5.4}$	$\underline{79.0}$
LoHA	$\underline{69.2_{2.3}}$	$89.4_{0.6}$	$89.1_{0.5}$	$\underline{57.3_{0.7}}$	$82.7_{5.7}$	77.5
VeRA	$66.2_{1.5}$	$87.6_{1.3}$	$84.8_{0.9}$	$49.2_{1.1}$	$87.5_{3.1}$	75.1
CLAM-2†	$64.7_{7.9}$	$\underline{90.5_{1.0}}$	$\underline{89.5_{0.9}}$	$57.2_{1.8}$	$\mathbf{92.9_{3.1}}$	$\underline{79.0}$
CLAM-3†	$68.2_{3.4}$	$\mathbf{91.3_{0.3}}$	$\mathbf{89.7_{0.2}}$	$\mathbf{58.6_{1.3}}$	$89.3_{4.7}$	**79.4**

For example, consider the length 3 chain deemed optimal for STS-B which contains (IA)^3 then LoHA then VeRA. The recommended learning rates provided in the papers are 3e-3 (Liu et al., 2022), $10^{-1.5}$ (Hyeon-Woo et al., 2021), and 1e-2 (Kopiczko et al., 2024) respectively. We pose that a fixed learning rate for all 3 of these parameter groups is suboptimal for two reasons. First, trying to fit one uniform learning rate leads to worse training as the optimizer loses finer control over the parameter updates. Second, each parameter in a PEFT technique can be expressive in different ways, learning complex, unique dependencies over the training data. A uniform learning rate restricts the ability of these adapters to effectively learn these nuances, leading to degraded performance.

Table 12. CLAM configurations found by the PEFT ordering search with single per-task global learning rates.

Task	Method	PEFT 1	PEFT 2	PEFT 3	Global LR	Train params
RTE	CLAM-2†	(IA)^3	LoHA (16)	—	1e-2	3.67M
	CLAM-3†	LoHA (4)	VeRA (4)	LoHA (8)	1e-2	2.71M
MRPC	CLAM-2†	(IA)^3	(IA)^3	—	7e-3	0.26M
	CLAM-3†	(IA)^3	LoHA (64)	(IA)^3	4e-3	14.41M
STS-B	CLAM-2†	LoHA (32)	(IA)^3	—	4e-3	7.21M
	CLAM-3†	(IA)^3	LoHA (4)	VeRA (32)	7e-3	1.07M
CoLA	CLAM-2†	(IA)^3	LoHA (32)	—	4e-3	7.21M
	CLAM-3†	(IA)^3	VeRA (4)	LoHA (16)	5e-3	3.72M
CB	CLAM-2†	LoHA (64)	(IA)^3	—	7e-3	14.28M
	CLAM-3†	VeRA (8)	LoRA (32)	LoRA (4)	2e-3	4.04M

A.10. Staging Algorithms

Algorithm 1 Adapter Integration and Fine-Tuning Algorithm for Generic Transformer

```
Require: GenericTransformer $G$, chain of adapters $C = [A_1, A_2, \dots, A_n]$, where $A_i$ is either differentiable ($A_d$) or
    non-differentiable ($A_n$)
Ensure: $A_{quantize}$ appears at most once in $C$
 1: assert count$(A_{quantize}, C) \leq 1$
 2: Split $C$ into subchains $S = [S_1, S_2, \dots, S_m]$ using the regular expression $(A_n^* A_d^*)$
 3: for each subchain $S_i \in S$ do
 4:   if $S_i$ contains only non-differentiable adapters then
 5:     for each non-differentiable adapter $A_n \in S_i$ sequentially do
 6:       $G \leftarrow A_n(G)$ {Apply $A_n$ sequentially to modify the backbone of $G$}
 7:     end for
 8:   else
 9:     for each differentiable adapter $A_d \in S_i$ sequentially do
10:       $G \leftarrow A_d(G)$ {Apply $A_d$ sequentially to $G$}
11:     end for
12:     Fine-tune $G$ on the training data
13:     Merge all $A_d$ in $S_i$ into the backbone of $G$:
14:     $G \leftarrow$ Merge$(G, S_i)$
15:     if any adapter up to this point includes quantization, i.e., $\exists A_{quantize} \in \bigcup_{j=1}^{i} S_j$ then
16:       Ensure the backbone of $G$ remains quantized
17:     end if
18:   end if
19: end for
20: return GenericTransformer $G$
```

Figure 3. Algorithm for integrating and finetuning a chain of adapters in a Generic Transformer.

In this section, we discuss the algorithm that takes a chain of adapters and applies them to a transformer. First, we check to make sure quantization only happens once in the chain. Second, we split the chain according to the regular expression. We then apply each subchain to the model, finetune it, and then merge the differentiable adapters back into the model. Finally, we maintain quantization if it has been applied in that chain or any previous chain.

A.11. Programmatic Implementation

We now provide an overview of abstractions within CLAM. First, all transformers (imported via `transformers`) are represented by a $\mathcal{G}$: GenericTransformer type, which contains a mapping $\mathcal{G}.L$ between generic labels and the paths to corresponding target matrices in the transformer (e.g., {"query": [transformer_name.block[0].q_proj, transformer_name.block[1].q_proj, $\ldots$], $\ldots$}). This mapping is useful for the method:

$$\mathcal{G}.\text{inject\_adapter}(M, \lambda x.f_{\mathcal{A}}(x, \theta))$$

in which M is a list of labels to be wrapped (e.g., {"query", "value"}) and $\lambda x.f_{\mathcal{A}}(x, \theta)$ is the function that takes a specific target weight tensor x and returns it wrapped and replaced by the instantiated adapter $\mathcal{A}$ using hyperparameters θ. $\mathcal{G}$ also has various functions such as $\mathcal{G}.\text{finetune}(D)$ which will finetune the model and all of its adapters with the dataset D; similarly, $\mathcal{G}.\text{prune}(D)$ will prune the model if there are pruning adapters in $\mathcal{G}$. To enable parameter-efficient training, CLAM also defines the utility function mark_adapters_as_trainable($\mathcal{G}$.auto_model) which in turn only enables gradient-based updates for adapter parameters and not the pretrained backbone.

We contend that, so long as the layers at the paths in $\mathcal{G}.L$ are linear, CLAM supports chaining at that path. In this way, CLAM is versatile in its applications to transformers and potential future architectures. To understand why, we now describe core functionality for adapters. An adapter $\mathcal{A}$ is used in functions `get_underlying_weight`($\mathcal{A}$) and `get_underlying_bias`($\mathcal{A}$), which query the layer that the adapter wraps for a weight and bias representation. These functions are used in the implementation of `get_equivalent_weight`($\mathcal{A}$) and `get_equivalent_bias`($\mathcal{A}$), which return a weight and bias equivalent in type to the underlying representation, now adding any new $\mathcal{A}$-specific operations to the gradient graph of $\mathcal{A}$'s underlying representation.