Activated LoRA: Fine-tuned LLMs for Intrinsics

Kristjan Greenewald, Luis Lastras, David Cox
IBM Research

January 2025

Abstract

Low-Rank Adaptation (LoRA) has emerged as a highly efficient framework for finetuning the weights of large foundation models, and has become the go-to method for data-driven customization of LLMs. Despite the promise of highly customized behaviors and capabilities, switching between relevant LoRAs in a multiturn setting is highly inefficient, as the key-value (KV) cache of the entire turn history must be recomputed with the LoRA weights before generation can begin. To address this problem, we propose Activated LoRA (aLoRA), which modifies the LoRA framework to only adapt weights for the tokens in the sequence *after* the LoRA is invoked. This change crucially allows aLoRA to accept the base model's KV cache of the input string, meaning that aLoRA can be instantly activated whenever needed in a chain without recomputing the cache. This enables building what we call *intrinsics*, i.e. highly specialized models invoked to perform well-defined operations on portions of an input chain or conversation that otherwise uses the base model by default. We use aLoRA to train a set of intrinsics models, demonstrating competitive accuracy with standard LoRA while achieving significant inference benefits, and illustrating their potential in a simple RAG pipeline with safety checking, uncertainty quantification, and hallucination detection. The codebase is at `https://github.ibm.com/Kristjan-H-Greenewald/activated-lora/tree/main`.

1 Introduction

The rapid adoption of large language models (LLMs) has catalyzed significant advancements in natural language processing tasks, from text generation to knowledge extraction. However, adapting these models to specific tasks or domains often demands fine-tuning their immense parameter space, a process that is computationally expensive and difficult to scale. Low-Rank Adaptation (LoRA) has addressed these challenges by introducing a parameter-efficient method for fine-tuning, enabling highly customized model behavior without the need to retrain or modify the entire model. By optimizing a small subset of low-rank matrices, LoRA has emerged as a lightweight and effective alternative for task-specific customization, particularly for large foundation models. Yet, while

LoRA excels in static or single-task scenarios, it presents inefficiencies when applied in multiturn interactions, where dynamic switching between multiple LoRA configurations is required.

In conversational or multi-task pipelines, this inefficiency arises from the necessity to recompute the key-value (KV) cache for all prior tokens when switching between LoRA weights. This recalculation introduces latency and computational overhead, limiting LoRA's usability in scenarios where rapid transitions between specialized behaviors are essential. Addressing this limitation, we introduce Activated LoRA (aLoRA), a novel extension of the LoRA framework designed to dynamically adapt the model's weights for tokens encountered after activation. By decoupling the adaptation process from the need to recompute the input's KV cache, aLoRA facilitates instantaneous switching between specialized models—or "intrinsics"—while maintaining seamless interaction with the base model. This innovation not only reduces computational costs but also unlocks new possibilities for deploying modular, task-specific behaviors within complex workflows.

1.1 Background

The Attention Mechanism Recall that the attention mechanism in each attention layer in LLMs (see Figure 1 for a typical architecture) generally takes the form

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^\top}{\sqrt{d_k}}\right)V, \tag{1}$$

where d_k is the dimension and Q, K, V are concatenated queries, keys, and values for the input tokens:

$$Q = XW^Q, \quad K = XW^K, \quad V = XW^V \tag{2}$$

where W^Q, W^K, and W^V are weight matrices.

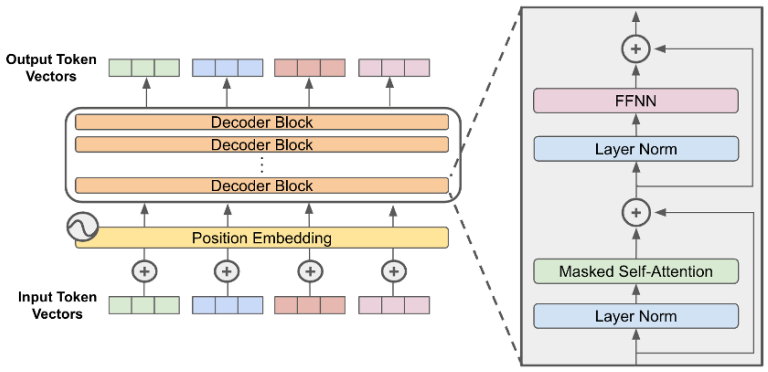

Figure 1: A generic LLM architecture.

LLM inference and the KV cache LLMs generate tokens autoregressively one at a time. While we do not write out the full equations here, note that generating this next token only requires computing the last row of this attention matrix if the keys and values of all prior tokens are precomputed (e.g. from when they were generated themselves or prefilled). This is because the last row of the attention matrix only uses the last row of Q, but the entire K and V matrices. These precomputed K and V values are called the KV cache. This observation creates significant speedups for LLM inference and is the key to all modern LLM inference engines.

LoRA LoRA adapts the attention weights W^Q, W^K, and W^V by replacing them with $W^Q + \Delta^Q$, $W^K + \Delta^K$, and $W^V + \Delta^V$, where $\Delta^Q, \Delta^K, \Delta_V$ are rank r matrices. This yields

$$Q = X(W^Q + \Delta^Q), \quad K = X(W^K + \Delta^K), \quad V = X(W^V + \Delta^V). \tag{3}$$

This lowers the number of parameters, making finetuning significantly more efficient Hu et al. [2021]. If the LoRA is active for the entire chain, then the KV cache-based inference applies and generation is efficient. If, on the other hand, any part of the input was generated or prefilled by the base model or another LoRA, then the KV cache for the input must be recomputed.

Intrinsics and the Post-Operator Framework We are motivated by what we call *intrinsics*, i.e. models that take in a structured context, followed by an invocation sequence that calls the required operation, and output an answer. Figure 3 illustrates this format. Note that implementing intrinsics as finetuned models would require the ability to *activate* them only when needed, which is not possible with standard LoRA without frequent recomputation of the KV cache.

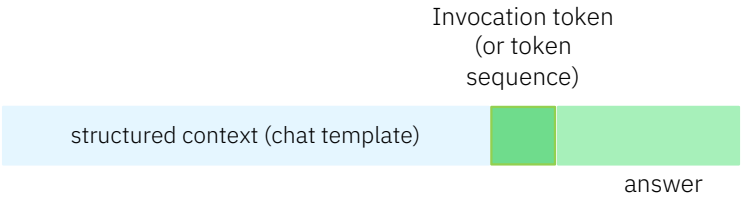

Figure 2: Post-operator framework for intrinsics. The aLoRA architecture is designed to adapt weights only on the green tokens, allowing it to reuse the base model cache for the blue input tokens.

In the following section, we will address this shortcoming, presenting our aLoRA method.

2 Activated LoRA

Our **Activated LoRA** (aLoRA) framework addresses this shortcoming. Just as in LoRA, aLoRA adapts the attention weights W^Q, W^K, and W^V by replacing them with $W^Q + \Delta^Q$, $W^K + \Delta^K$, and $W^V + \Delta^V$, where $\Delta^Q, \Delta^K, \Delta_V$ are rank r matrices. The difference lies in how these adapted weights are used. We assume that the the default generation model for the chat is the base model, and that intrinsics only operate on these base model generations. As a result, we can assume that the base model has precomputed a KV cache for the input context (the blue region in Figure 2).

The aLoRA architecture is designed to **directly reuse this base model KV cache**. In the attention mechanism (1), aLoRA only adapts the Q, K, V matrices for tokens occuring after the start of the invocation sequence. Specifically, instead of (3) we have

$$Q = \begin{bmatrix} X_{before}W^Q \\ X_{after}(W^Q + \Delta^Q) \end{bmatrix}, \quad K = \begin{bmatrix} X_{before}W^K \\ X_{after}(W^K + \Delta^K) \end{bmatrix}, \quad V = \begin{bmatrix} X_{before}W^V \\ X_{after}(W^V + \Delta^V) \end{bmatrix}, \tag{4}$$

where X_{before} and X_{after} are the portions of X coming before and after the aLoRA model is invoked. Note that if X_{before} is associated with tokens generated or prefilled by the base model, then $X_{before}W^K$ and $X_{before}W^V$ are already in the KV cache and do not need to be recomputed. Similarly, any tokens that have been prefilled or generated by the aLoRA model have keys and values processed by the adapted weights, so $X_{after}(W^K + \Delta^K)$ and $X_{after}(W^V + \Delta^V)$ are also available (except for the current token being generated). As a result, the aLoRA architecture can seemlessly reuse the existing base model KV cache as well as continue to maintain its own KV cache as it generates.

Note that the post-operator framework for intrinsics is highly amenable to supervised finetuning of aLoRA via data collators, and furthermore ensures that the tokens affected by the aLoRA are isolated to only the answer. As a result, once the aLoRA is turned off and the base model resumes generation, the base model can either resume from its own cache where the aLoRA began, or prefill the (usually short) answer from the aLoRA if needed.

2.1 Invocation

We found it useful to demarcate the activation of the aLoRA adapter via an *invocation token sequence*. While the aLoRA will often be invoked programmatically by the runtime, this design in principle allows for the base model (or other intrinsics) to call the aLoRA model themselves. In our current implementation, except for the 1-token experiments below, the aLoRA weights are activated at the *start* of the invocation sequence, since it is typically inserted by the runtime and may not have a KV cache precomputed. Note that the invocation sequence is optional in principle, i.e. it can be an empty string.

The invocation sequence has several additional benefits. The invocation sequence can be designed to

- Conform to the chat template (for instance by making the aLoRA response its own turn with a specialized role)
- Provide a short prompt to aid the learning process (the name of the intrinsic may be enough)
- Give the adapter weights more tokens to process the input prior to generating the output, possibly improving performance.

2.2 Training and Inference

Our implementation of aLoRA at `https://github.ibm.com/Kristjan-H-Greenewald/activated-lora/tree/main` is designed to seamlessly use standard Huggingface libraries for training and inference (using the KV cache).

Training data is specified as a set of (possibly multiturn) inputs and single-turn completions, typically with the base model's chat template applied. An invocation token sequence for the aLoRA model is optionally specified as described in the previous section. This invocation sequence is appended to the input sequence, and the model is finetuned to produce the output given the input.

To train the aLoRA adapter, a base model is specified, and we permit low-rank adapters to be applied to any (or all) query, key, and value blocks in the attention layers.[1] In training, the aLoRA is aware of which tokens occur before the invocation sequence, and does not adapt the weights for those tokens (as in (4)).

Generation with aLoRA is inherently simple due to the structure (4). When generating with the cache, the model simply uses the concatenation of the input base model KV cache with its own cache.

3 (Initial) Results

In this section, we trained aLoRA adapters for IBM's Granite 3.1 8b Instruct model. All attention weights (keys, queries, values) were adapted in all layers, using rank 32 adapters.

Increased rank for aLoRA Note that the rank we used (32) is higher than would be needed with standard LoRA adapters, which for these tasks only needed to be rank 4. We observed that raising the rank in this manner is generally necessary for aLoRA adapters, and offer the following intuition for this. LoRA adapters are free to adapt the weights for the keys and values for tokens prior to activation, so they are able to "compress" information needed for generation into the low-rank signal captured by the adaptation. This signal can then be "picked up" by the adapted query weights for generated tokens. On the other hand, aLoRA adapters are not able to do this compression, only

[1] We do not currently support adapting any other blocks.

being able to access base model KVs with adapted query weights. Hence, if the query adaptation is too low rank, it is not able span enough of the base model keys to capture all needed information. As a result, higher rank adaptations may be crucial. We plan to empirically explore this further.

3.1 Intrinsics Tasks and Data

To test the aLoRA framework, we consider three specific intrinsics tasks previously appearing in Greenewald et al. [2024]. See the appendix for the datasets used for each task.

Uncertainty Quantification This intrinsic is designed to provide a Certainty score for model responses to user questions. The model will respond with a number from 0 to 9, corresponding to 5%, 15%, 25%,...95% confidence respectively. Training data for these confidence scores are obtained by applying the UQ calibration method of Shen et al. [2024] to a large, diverse set of benchmark question answering datasets (see appendix) and quantizing the predicted confidences. These percentages are *calibrated* in the following sense: given a set of answers assigned a certainty score of X%, approximately X% of these answers should be correct.

Safety Checking The Safety Intrinsic is designed to raise an exception when the user query is unsafe.[2] The model responds with 'Y' (safe), and 'N' otherwise.

Hallucination Detection The Hallucination Detection intrinsic is designed to detect when an assistant response to a user question with supporting documents is not supported by those documents. Response with a 'Y' indicates hallucination, and 'N' no hallucination.

3.2 Single-token tests

Baselines As a baseline, we consider individual (standard) LoRAs. Here, each of the three intrinsics has its own LoRA. As these cannot re-use the cache, deploying this approach is not desirable but serves as an accuracy benchmark to ensure aLoRA does not lose too much performance.

Metrics The Safety and Hallucination intrinsics are essentially binary classification tasks, hence we compute classification error rate on a held-out test set (200 samples per task). The Uncertainty Quantification intrinsic returns an ordinal score, so we compute the mean absolute error between the predicted integer and the target integer.

Results are shown in Figure 3. Note that performance is largely unchanged using aLoRA instead of standard LoRA.

[2]Training data considers violence, threats, sexual and explicit content and requests to obtain private identifiable information.

	LoRA	aLoRA (ours)
Uncertainty Quantification	0.50 MAE	0.49 MAE
Safety	1.6%	2.25%
Hallucination Detection	21.0%	22.0%

Figure 3: Comparison between LoRA and aLoRA test error for three intrinsics, both in the single-token output regime. Note that aLoRA does not lose meaningful performance.

3.3 Multi-token tests

To test aLoRA's ability to learn to generate multi-token responses, we modified the Uncertainty Quantification and Safety intrinsics to have the following multitoken output:

- Uncertainty Quantification: X5% (3 tokens)
- Safety: Either "Yes, this is a safe prompt." or "No, this is unsafe!"

Results are shown in Figure 4. The same performance metrics are used as in the single token case, with a slight improvement for UQ. Perhaps unsurprisingly given the increase in task complexity, the multitoken safety output degrades performance slightly but remains high. Note further that additional hyperparameter tuning on the multi-token output could improve performance as well.

	1-token output	Multi-token output
Uncertainty Quantification	0.485 MAE	0.405 MAE
Safety	2.25%	4.25%

Figure 4: UQ MAE and Safety classification error rates for aLoRA in the single-token and multi-token output regimes.

4 Conclusion

n this work, we presented Activated LoRA (aLoRA), a novel extension of the LoRA framework that enables efficient and dynamic adaptation of large language models without requiring the recomputation of the key-value (KV) cache. By modifying LoRA to adapt weights only for tokens generated after activation, aLoRA facilitates seamless integration into multiturn settings, enabling the use of specialized "intrinsics" for well-defined operations within a broader conversational or processing pipeline.

Our experiments demonstrate that aLoRA maintains competitive accuracy compared to standard LoRA while significantly reducing inference costs. This capability was showcased through applications in uncertainty quantification,

safety checking, and hallucination detection, all within a retrieval-augmented generation (RAG) framework. The flexibility and efficiency of aLoRA highlight its potential to streamline the deployment of modular, task-specific models in complex workflows, paving the way for more adaptive and responsive LLM-driven systems. Future work will explore integrating aLoRA into modern inference servers such as vLLM and developing expanded use cases for aLoRA.

Acknowledgments

The authors would like to thank Nathalie Baracaldo and Chulaka Gunasekara for providing data for and defining the Safety and Hallucination Detection intrinsics, respectively (previously appearing in Greenewald et al. [2024]). We also thank Jiacheng Zhu and Mikhail Yurochkin for early conversations regarding this work.

References

Swapnaja Achintalwar, Adriana Alvarado Garcia, Ateret Anaby-Tavor, Ioana Baldini, Sara E Berger, Bishwaranjan Bhattacharjee, Djallel Bouneffouf, Subhajit Chaudhury, Pin-Yu Chen, Lamogha Chiazor, et al. Detectors for safe and reliable llms: Implementations, uses, and limitations. *arXiv preprint arXiv:2403.06009*, 2024.

IBM Granite Team. Granite 3.0 language models, 2024.

Kristjan Greenewald, Nathalie Baracaldo, Chulaka Gunasekara, Lucian Popa, and Mandana Vaziri. Granite-3.0-8b-lora-intrinsics-v0.1, 2024. URL `https://huggingface.co/ibm-granite/granite-3.0-8b-lora-intrinsics-v0.1`.

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

Maohao Shen, Subhro Das, Kristjan Greenewald, Prasanna Sattigeri, Gregory Wornell, and Soumya Ghosh. Thermometer: Towards universal calibration for large language models. *arXiv preprint arXiv:2403.08819*, 2024.

A Training datasets for intrinsics

A.1 QA datasets for Uncertainty Quantification Intrinsic

The following datasets were used for calibration and/or finetuning.

- BigBench

- MRQA
- newsqa
- trivia_qa
- search_qa
- openbookqa
- web_questions
- smiles-qa
- orca-math
- ARC-Easy
- commonsense_qa
- social_i_qa
- super_glue
- figqa
- riddle_sense
- ag_news
- medmcqa
- dream
- codah
- piqa

A.2 Training data for Safety Intrinsic

The following public datasets were used for finetuning.

- yahma/alpaca-cleaned
- nvidia/Aegis-AI-Content-Safety-Dataset-1.0
- A subset of Anthropic/hh-rlhf
- Ibm/AttaQ
- google/civil_comments
- allenai/social_bias_frames

A.3 Training data for Hallucination Detection Intrinsic

The following public datasets were used for finetuning. The details of data creation for RAG response generation are available in the Granite Technical Report Granite Team [2024]. For creating the hallucination labels for responses, the technique described in Achintalwar et al. [2024] was used.

- MultiDoc2Dial
- QuAC