ACTIVATED LORA: FINE-TUNED LLMS FOR INTRINSICS

Kristjan Greenewald, Luis Lastras, Thomas Parnell, Vraj Shah, Lucian Popa, Giulio Zizzo, Chulaka Gunasekara, Ambrish Rawat, David Cox
IBM Research

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 aLoRA 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. The codebase is at `https://github.com/IBM/activated-lora`.

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)Hu et al. (2021) has addressed these challenges by introducing a parameter-efficient method for fine-tuning Houlsby et al. (2019), 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 agentic or reasoning pipelines.

This inefficiency arises from the necessity to recompute the representation of all tokens prior to generation when switching between LoRA weights. In the case of the popular attention mechanism introduced by Vaswani et al. (2017), such a representation takes the form of the key-value (KV) cache of such prior tokens, but more generally, it is a problem that reoccurs even in different architectures; for example, in a state-space model, the state representation of the tokens prior to generation would need to be recomputed if the matrices A and B are updated by a low rank correction. This recalculation introduces significant latency, GPU memory, and computational overhead that all scale with the length of the context that must be prefilled. This limits LoRA's usability in scenarios where rapid transitions between specialized behaviors are essential.

If only a few specialized behaviors are needed, all needed tasks can in principle be trained into the LoRA while retaining as much of the base model behavior as possible (here the LoRA would be used for all parts of the pipeline), but this solution has major limitations. In particular, by modifying the weights of the model for model calls that are best handled by the base model, performance degradation is likely on those tasks. Additionally, multi-task LoRA training is significantly more difficult than training a LoRA for a single task, and this approach is inherently non-modular, requiring complete retraining to incorporate new intrinsics abilities.

Addressing this inability to reuse the base model representation of the tokens prior to generation, 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 representation, 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 highly modular, task-specific behaviors within complex workflows.

This article is a sister companion to Danilevsky et al. (2025), which focuses on the development of the RAG intrinsics for which we provide corresponding aLoRA implementations.

1.1 Background

We will illustrate the concept behind activated LoRA in the context of the ubiquitous attention mechanism introduced by Vaswani et al. (2017).

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.

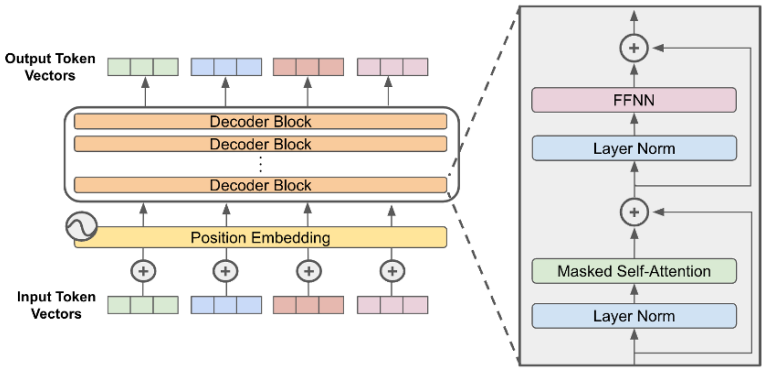


Figure 1: A generic LLM architecture.

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.

2 INTRINSICS AND LATE PROMPTS

Within the discipline of software engineering, *intrinsics* are generally useful functions that are built into a programming language whose implementation can be optimized by a compiler. We define an LLM intrinsic to be a capability that can be invoked through a well defined API that is reasonably stable and independent across model generations and families of how the intrinsic is implemented. Performance metrics such as accuracy, latency and throughput may vary significantly across such implementations.

Inference on LLMs is becoming quite sophisticated; for example, sending prompts directly to LLMs is a largely deprecated practice by developers, with structured interfaces (e.g. tool descriptions, and a sequence of messages from a variety of roles and with various content types) being processed by lower level, model specific software or template configurations to translate such structured interface to actual tokens. Thus we posit LLM intrinsics are best implemented by the combination of a model and a software layer that leverages advances on model inference technologies to provide differentiated performance through a stable interface.

The concept of activated LoRA takes an opinionated view on how such differentiated performance could be attained. As inspiration, we note that instructions in LLMs can appear in many places in a prompt; Figure 2 illustrates an example of two such places: an "early prompting" case where the instruction precedes the content, or a "late prompting" case where the content precedes the instruction. In the latter, the instruction does not have to be revealed ahead of time, making the representation of the content potentially useful for many tasks. We note that these paradigms can take on many shapes; for example, early prompting could just as well be substituted by prefix tuning methods such as Li and Liang (2021) with the same overall outcome.

The reader may appreciate that a LoRA adapter, like the prefix tuning example above, shares some similarities with the early prompting concept; the LLM's internal representation of the content is fit-for-purpose, specific to the LoRA adapter and not easily reusable. Late prompts, on the other hand, suffer from a potential disadvantage compared to early prompts since there is no opportunity for the content to be represented with the specific task in mind. Can we find a LoRA-like techology, that gives us the benefits of late prompts with none of the potential accuracy disadvantages?

In the following section, we will address this shortcoming, presenting our Activated LoRA method, which *activates* adapted weights only on tokens corresponding to the intrinsic instruction and generation.

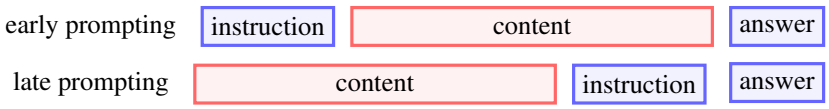


Figure 2: Late vs. early prompting framework for intrinsics. The aLoRA adapter architecture is designed to preserve the cache-reuse benefits of *late prompting* by adapting weights only on the blue tokens, allowing it to reuse the base model cache for the red input tokens.

3 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 occurring after the start of the invocation sequence. Specifically, instead of (3) we have

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

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 late prompting 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.

3.1 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.

3.2 Comparison to LoRA at inference time

In Figure 3, an example is shown that conforms to the late prompting framework. As part of an ongoing base model chat, a user query comes in, and (optionally) documents or other information is also included. The base model then generates a response. We wish to use Certainty and Hallucination intrinsics to check this base model response, so we append the corresponding invocation sequences and generate the intrinsics outputs. These intrinsics calls can be done in parallel as they do not depend on each other. For the LoRA-based intrinsics, the entire context must be prefilled using the LoRA model prior to being able to generate the intrinsic output. On the other hand, Activated LoRA can reuse the base model cache for the vast majority of the input, only needing to prefill its own short prompt. This results in signficant latency gains.

3.3 Training and Inference

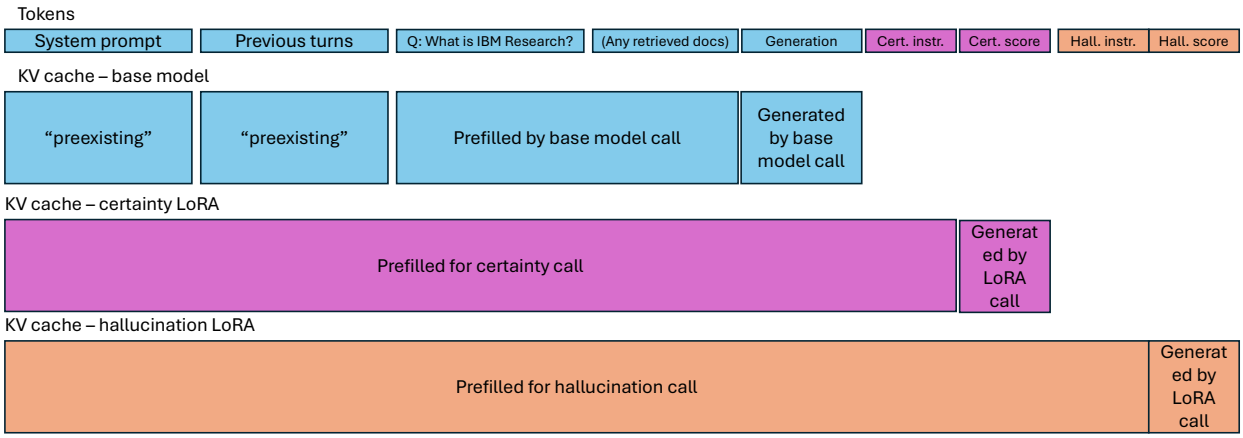
Our implementation of aLoRA at `https://github.ibm.com/Kristjan-H-Greenewald/activated-lora/tree/main` is designed to seamlessly use standard Huggingface libraries Wolf et al. (2019) for training and inference (using the KV cache).

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.

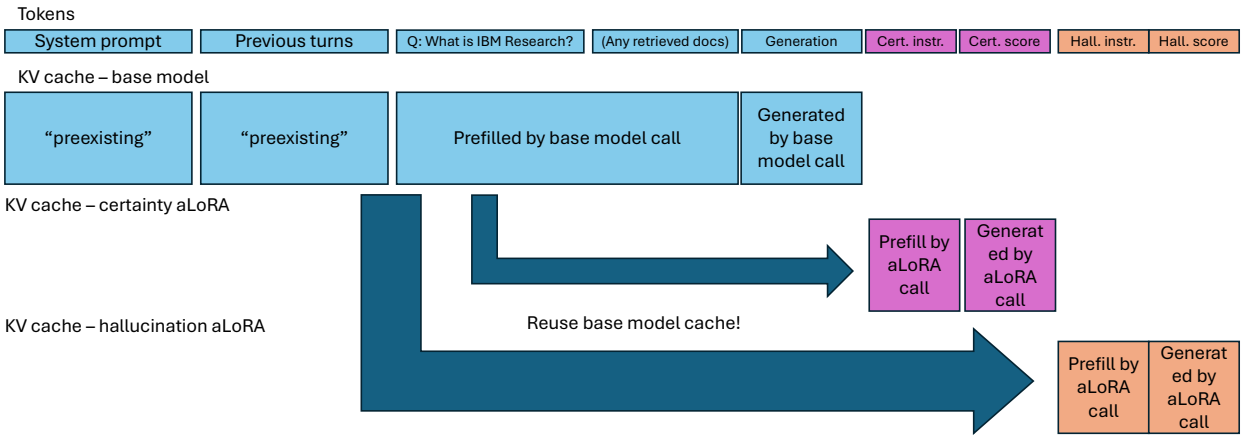
To train the aLoRA adapter, a base model is specified, and we permit low-rank adapters to be applied to any (or all) query, key, and value blocks in the attention layers.[1] In training, the aLoRA is aware of which tokens occur before the invocation sequence, and does not adapt the weights for those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

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



(a)



(b)

Figure 3: LoRA (a) vs. aLoRA (b) intrinsic generation calls, checking an output generated by the base model. Note that aLoRA can reuse the long base model cache, saving signficant prefill costs in both compute and GPU memory.

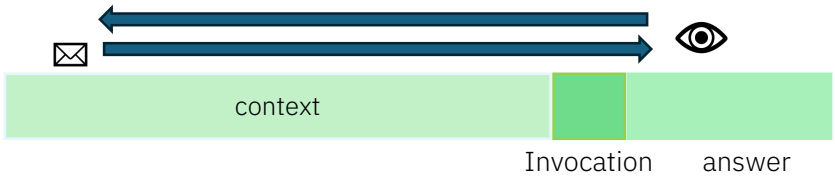
3.4 Model Capacity and Increased Rank

In our experiments, we universally observed that aLoRA adapters required higher rank than LoRA adapters to get good performance. We 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 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. Figure 4 illustrates this. In our experiments, rank of 32 seems to be sufficient in most cases, which is still vastly smaller than the size of the base model. Whenever comparing to LoRA models, we chose a LoRA rank that achieved top performance for LoRA, rather than attempting to match the ranks between LoRA and aLoRA.

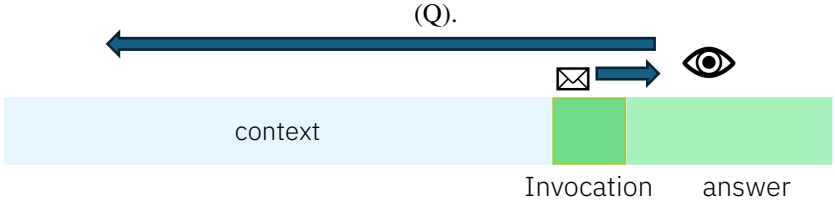
Note that the above provides the key motivation for starting the adapted weights at the intrinsic instruction tokens, rather than waiting to activate the weights only at generation time—this choice boosts model capacity.

4 Training adapters

In this section, we trained aLoRA adapters for IBM’s Granite 3.2 8b Instruct model, and compared their performance to performance achieved on trained LoRA models. As the LoRA models cannot re-use the base



(a) LoRA is able to modify the (K)Vs of the context to send (new) messages forward to the generation queries (Q).



(b) aLoRA is not able to do this (for the context), it can only look back at it with queries (Q).

Figure 4: ALoRA model capacity losses due to not being able to modify the keys and values of input tokens prior to the invocation sequence to send "messages" forward to the generation step. We will see empirically that any performance losses can be eliminated by increasing the rank of the adapter.

model cache, deploying them is not efficient, but serves as an accuracy benchmark to ensure aLoRA does not lose too much performance.

For aLoRA, all attention weights (keys, queries, values) were adapted in all layers, using rank 32 adapters. For both LoRA and aLoRA, the learning rate and number of epochs were tuned to achieve the best validation performance.

Note that since all the below models use the same base model (IBM's Granite 3.2 8b Instruct), they can be swapped in and out as needed in *the same flow*. The reader can envision the possibilities of what can be built with these intrinsics, e.g. for RAG. See Danilevsky et al. (2025) for an exploration of composing these intrinsics (using LoRAs).

Comparing LoRA and aLoRA To test the aLoRA framework, we consider several specific intrinsics tasks. Most of these intrinsics tasks were proposed or discussed in Danilevsky et al. (2025) , see that paper for deeper motivation for each intrinsic and extensive experimental results comparing standard LoRA implementations to other baselines. In this work, we therefore focus entirely on the comparison between LoRA and aLoRA performance, aiming to show that aLoRA models do not see meaningful degradation vs. LoRA models, while obtaining significant runtime benefits as discussed above.

See the appendices for additional details for the tasks and intrinsics.

4.1 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. (2024a) to a large, diverse set of benchmark question answering datasets (see appendix) and quantizing the predicted confidences. Further details on the training process are in the appendix.

Certainty score interpretation The returned 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. Here "approximately" can be quantified via the expected calibration error, or ECE. Essentially what happens is teaching the adapter model what the base model knows and doesn't know. This inherently requires generalization to questions of wildly varying difficulty (some of which may be trick questions!) and to settings not in training. Intuitively, it does this by extrapolating based on related questions it has been evaluated on in training - this is an inherently inexact process and leads to some hedging.

Model	Unans. Precision	Unans. Recall	Unans. F1	Ans. Precision	Ans. Recall	Ans. F1	Weighted F1
Granite 3.2-8b LoRA	84.2	68.0	75.2	73.1	87.2	79.5	77.4
Granite 3.2-8b aLoRA	83	81.1	82	81.4	83.3	82.4	82.2

Table 1: Comparison of classification performance across models on SQUADRUN Dev set. Metrics are broken down by class (Answerable vs. Unanswerable) and include precision, recall, and F1 score.

Usage Note that the model is evaluating responses from its base model - in other words, it cannot be applied to generations from other models. The aLoRA architecture is thus particularly well-suited to this use case. In practice, the goal is to provide a highly efficient uncertainty score without having to resort to expensive larger judge models.

Metrics The Uncertainty Quantification intrinsic returns an ordinal score, so we compute the mean absolute error between the predicted integer and the target integer in the training data.

Performance Results Results are shown in Figure 5. Note that performance is largely unchanged using aLoRA instead of standard LoRA.

	LoRA	aLoRA
Uncertainty Quantification	0.50 MAE	0.49 MAE

Figure 5: Comparison between LoRA and aLoRA test error for the Uncertainty Quantification intrinsic. Note that aLoRA does not lose meaningful performance.

4.2 Answerability Determination

This intrinsic is designed to assess whether a user's final query in a multi-turn conversation can be answered given the retrieved documents. The model takes as input the full conversation history and a set of documents, and classifies the query as answerable or unanswerable based solely on the information contained in the set of input documents. This is useful in RAG settings. For instance, this intrinsic can help decide whether to proceed with generation or abstain with an "I don't know" response.

The input to the model is a list of conversational turns and a list of documents converted to a string using `apply_chat_template` function. These turns can alternate between the user and assistant roles. The last turn is from the user. The list of documents is a dictionary with text field, which contains the text of the corresponding document. To prompt the aLoRA adapter to determine answerability, a special answerability role is used to trigger this capability of the model. The role includes the keyword `"answerability": <|start_of_role|>answerability<|end_of_role|>` When prompted with the above input, the model generates the answerable or unanswerable output. See Danilevsky et al. (2025) for more details.

Training Details The aLoRA and LoRA adapters were fine-tuned using PEFT under the following regime: rank = 32, learning rate = 5e-6, number of epochs = 25, with early stopping based on validation set, and 90/10 split between training and validation.

Evaluation: Answerability Classification Performance We evaluated aLoRA model against LoRA and baselines on binary answerability classification using two separate benchmarks:

- Single-turn Setting (SQUADRun Benchmark Rajpurkar et al. (2018)): In this setting, the user query and the supporting documents are provided. Our model was evaluated against standard baselines to measure its ability to determine whether a standalone question is answerable based on the document set. Table 1 shows the classification results.
- Multi-turn Setting (MT-RAG Benchmark Katsis et al. (2025)): In this setting, the model is given the full multi-turn conversation history along with the supporting documents. This benchmark evaluates the model's ability to assess answerability when the final user query can also depend on prior turns for context. Table 2 shows the results.

Model	Unans. Precision	Unans. Recall	Unans. F1	Ans. Precision	Ans. Recall	Ans. F1	Weighted F1
Granite 3.2-8b LoRA	85.4	89.3	87.3	87.0	82.4	84.6	86.1
Granite 3.2-8b aLoRA	85.8	89.1	87.4	86.8	83	84.9	86.2

Table 2: Comparison of classification performance across models on MT-RAG Benchmark. Metrics are broken down by class (Answerable vs. Unanswerable) and include precision, recall, and F1 score.

Overall, the aLoRA model does not lose performance relative to the LoRA model, in fact, aLoRA achieves the best classification performance on SQUADRun. See Danilevsky et al. (2025) for additional baselines.

4.3 Query Rewrite

This intrinsic is generally applicable for multi-turn conversational use cases, and its role is to perform rewrites of user queries for better performance for the downstream tasks. It is especially useful in RAG settings, see the metrics and evaluation results below. The query rewrite task is as follows: given a multi-turn conversation between a user and an AI assistant, de-contextualize the last user utterance (query) by rewriting it (whenever necessary) into an equivalent version that is standalone and can be understood by itself.

The rewrite is typically an expansion that in-lines, into the query, any implicit references that are made to entities, concepts, or even parts of the conversation that occur in the previous turns (either by the user or the AI assistant). Such expansion can include co-reference resolution (i.e., replacement of pronouns with the actual entities), handling of ellipsis, which is the common linguistic phenomenon where parts of a sentence or phrase are omitted by the user, but can be understood from the context (i.e., for whom, of what, with respect to something discussed above, etc.). The rewritten query can be sent to downstream tasks (e.g., to a retriever in a RAG setting) as a better replacement for the original user query, and without the need for (a potentially very long) context.

The input to the model is a list of conversational turns converted to a string using `apply_chat_template` function. These turns can alternate between the user and assistant roles, and the last turn is assumed to be from the user. To prompt the aLoRA adapter to rewrite the last user turn, a special rewrite role is used to trigger the rewrite capability of the model. The role includes the keyword "rewrite" followed by a short description of the query rewrite task.

Note: Even though one main application for query rewrite is in RAG settings, this intrinsic can be used to rewrite user questions for other conversational use cases (e.g., to access a database, or other APIs, or tools). As such, the adapter does not need any RAG documents (that may be present in the context, in a RAG setting) and uses only the dialog turns with what is being said between the user and assistant.

Training Both the aLoRA and LoRA adapters were fine-tuned under the following regime: rank = 32, number of epochs = 25, with early stopping based on validation set, and 90/10 split between training and validation.

4.3.1 Query Rewrite: Evaluation on the Retrieval Task

We evaluate `Recall@k` on the MT-RAG benchmark Katsis et al. (2025), using either LoRA or aLoRA models to rewrite the query for the retriever. All retrieved passages are obtained using the Elser retriever with the same settings as in the above paper. We evaluate on three different testsets: a) full MT-RAG dataset (842 data points with last user turns); b) the non-standalone subset of MT-RAG dataset, which is a subset of 260 (out of 842) last user turns that were annotated by humans as non-standalone (i.e., they are dependent on the prior context); c) the standalone subset of MT-RAG dataset, which is the complementary subset, with all the last user turns that were annotated by humans as standalone.

Retrieval recall evaluation (Recall@k) with different query rewrite strategies, evaluated on full, non-standalone and standalone subsets of MT-RAG dataset are shown in Tables 3, 4, and 5 respectively.

Rewrite Strategy	Recall@5	Recall@10	Recall@20
aLoRA	0.54	0.66	0.74
LoRA	0.56	0.68	0.76

Table 3: Query rewrite strategies on the retrieval task of full MT-RAG dataset

Rewrite Strategy	Recall@5	Recall@10	Recall@20
aLoRA	0.42	0.54	0.64
LoRA	0.44	0.57	0.66

Table 4: Query rewrite strategies on the retrieval task of non-standalone subset of MT-RAG

Rewrite Strategy	Recall@5	Recall@10	Recall@20
aLoRA	0.63	0.75	0.82
LoRA	0.63	0.75	0.83

Table 5: Query rewrite strategies on the retrieval task of standalone subset of MT-RAG

Note that throughout, performance numbers for aLoRA and LoRA are within a point or two. See CITERAG for additional comparisons showing that these approaches outperform benchmarks and are very close to the performance with gold rewrites.

4.3.2 Query Rewrite: Evaluation on Answer Generation

We evaluate answer generation quality, with top-k passages retrieved under the various query rewrite strategies for the retriever. We choose here $k = 20$, but similar trends take place for other values of k. We used Granite-3.2-8b instruct as the answer generator, and RAGAS Faithfulness (RAGAS-F) and RAD-Bench score as metrics for answer quality. We use the same three testsets as above.

The answer quality evaluation using RAGAS-F and RAD-Bench on full, non-standalone and standalone subsets of MT-RAG dataset are shown in Tables 6, 7, and 8 respectively.

Rewrite Strategy	RAGAS-F	RAD-Bench
aLoRA	0.81	0.69
LoRA	0.81	0.70

Table 6: Impact of query rewrite strategies on answer generation on full MT-RAG dataset

Rewrite Strategy	RAGAS-F	RAD-Bench
aLoRA	0.77	0.69
LoRA	0.79	0.69

Table 7: Impact of query rewrite strategies on answer generation on non-standalone subset of MT-RAG

Rewrite Strategy	RAGAS-F	RAD-Bench
aLoRA	0.83	0.70
LoRA	0.83	0.71

Table 8: Impact of query rewrite strategies on answer generation on standalone subset of MT-RAG

As with Recall, observe that LoRA and aLoRA performance is very close, indicating that moving to aLoRA for inference reasons does not require giving up any performance.

4.4 JAILBREAK DETECTION

This intrinsic is designed for detecting jailbreak risk within user prompts. Prompts with jailbreak risk vary across a wide range of attack styles - from direct instructions, to encoding-style, social-hacking based attacks and even ones that exploit special token or context overload (Rawat et al., 2024). In our experiments we focused on training intrinsics for detecting social hacking style of adversarial prompts. The intrinsic is trained to return a binary label - "Y" indicating jailbreak risk present and "N" indicating no risk, when invoked. We evaluate jailbreak intrinsic across a mixture of samples with jailbreak risk (Shen et al., 2024b; Lin et al., 2023; Wan et al., 2024) and benign samples (Conover et al., 2023). As with the other intrinsics, we see very little performance difference between LoRA and aLoRA.

Jailbreak Risk Detector	**Acc**	**TPR**	**FPR**
aLoRA	0.925	0.863	0.013
LoRA	0.944	0.898	0.009

Table 9: Performance of jailbreak risk detectors across an aggregated mixture of prompts with jailbreak risk and benign samples.

5 CONCLUSION

In 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, answerability determination, hallucination detection, query rewrite, and jailbreak detection. 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.

Next steps Future work will explore integrating aLoRA into modern inference servers such as vLLM and developing expanded use cases for aLoRA models in agentic and test-time-scaling applications.

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.

Mike Conover, Matt Hayes, Ankit Mathur, Jianwei Xie, Jun Wan, Sam Shah, Ali Ghodsi, Patrick Wendell, Matei Zaharia, and Reynold Xin. Free dolly: Introducing the world's first truly open instruction-tuned llm, 2023. URL `https://www.databricks.com/blog/2023/04/12/dolly-first-open-commercially-viable-instruction-tuned-llm`.

Marina Danilevsky, Kristjan Greenewald, Chulaka Gunasekara, Maeda Hanafi, Lihong He, Yannis Katsis, Krishnateja Killamsetty, Yatin Nandwani, Lucian Popa, Dinesh Raghu, Frederick Reiss, Vraj Shah, Khoi-Nguyen Tran, Huaiyu Zhu, and Luis Lastras. A library of llm intrinsics for retrieval-augmented generation, 2025.

IBM Granite Team. Granite 3.0 language models, 2024.

Neil Houlsby, Andrei Giurgiu, Stanislaw Jastrzebski, Marco Morrone, and Quentin de Laroussilhe. Parameter-efficient transfer learning for nlp. In *International Conference on Machine Learning*, pages 2790–2799, 2019.

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.

Yannis Katsis, Sara Rosenthal, Kshitij Fadnis, Chulaka Gunasekara, Young-Suk Lee, Lucian Popa, Vraj Shah, Huaiyu Zhu, Danish Contractor, and Marina Danilevsky. MTRAG: A multi-turn conversational benchmark for evaluating retrieval-augmented generation systems, 2025. URL https://arxiv.org/abs/2501.03468.

Xi Li and Percy Liang. Prefix-tuning: Optimizing continuous prompts for generation. In *International Conference on Learning Representations*, 2021.

Zi Lin, Zihan Wang, Yongqi Tong, Yangkun Wang, Yuxin Guo, Yujia Wang, and Jingbo Shang. Toxicchat: Unveiling hidden challenges of toxicity detection in real-world user-ai conversation. In Houda Bouamor, Juan Pino, and Kalika Bali, editors, *Findings of the Association for Computational Linguistics: EMNLP 2023, Singapore, December 6-10, 2023*, pages 4694–4702. Association for Computational Linguistics, 2023. doi: 10.18653/V1/2023.FINDINGS-EMNLP.311. URL https://doi.org/10.18653/v1/2023.findings-emnlp.311.

Pranav Rajpurkar, Robin Jia, and Percy Liang. Know what you don't know: Unanswerable questions for SQuAD. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 784–789, 2018.

Ambrish Rawat, Stefan Schoepf, Giulio Zizzo, Giandomenico Cornacchia, Muhammad Zaid Hameed, Kieran Fraser, Erik Miehling, Beat Buesser, Elizabeth M. Daly, Mark Purcell, Prasanna Sattigeri, Pin-Yu Chen, and Kush R. Varshney. Attack atlas: A practitioner's perspective on challenges and pitfalls in red teaming genai. *CoRR*, abs/2409.15398, 2024. doi: 10.48550/ARXIV.2409.15398. URL https://doi.org/10.48550/arXiv.2409.15398.

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*, 2024a.

Xinyue Shen, Zeyuan Chen, Michael Backes, Yun Shen, and Yang Zhang. "do anything now": Characterizing and evaluating in-the-wild jailbreak prompts on large language models. In Bo Luo, Xiaojing Liao, Jun Xu, Engin Kirda, and David Lie, editors, *Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security, CCS 2024, Salt Lake City, UT, USA, October 14-18, 2024*, pages 1671–1685. ACM, 2024b. doi: 10.1145/3658644.3670388. URL https://doi.org/10.1145/3658644.3670388.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems*, pages 5998–6008, 2017.

Shengye Wan, Cyrus Nikolaidis, Daniel Song, David Molnar, James Crnkovich, Jayson Grace, Manish Bhatt, Sahana Chennabasappa, Spencer Whitman, Stephanie Ding, Vlad Ionescu, Yue Li, and Joshua Saxe. CYBERSECEVAL 3: Advancing the evaluation of cybersecurity risks and capabilities in large language models. *CoRR*, abs/2408.01605, 2024. doi: 10.48550/ARXIV.2408.01605. URL https://doi.org/10.48550/arXiv.2408.01605.

Thomas Wolf et al. Huggingface's transformers: State-of-the-art natural language processing. https://huggingface.co/transformers/, 2019.

A Additional details for intrinsics

A.1 Uncertainty Quantification

First, a probe-based model was trained to produce calibrated certainty scores, using a large diverse collection of QA datasets detailed in the next section. Note that throughout, the Granite chat template was used. The procedure for this was as follows:

1. A “meta-dataset" was created containing User inputs, Answer generations (from the Granite base model), and correctness labels for those generations.
2. For each row in the meta-datset, the base model was prompted with input of the form (User inputs, Answer generations, meta prompt), where the meta prompt was

   ```
   Is the above answer correct?\n <A> Yes, \n<B> No, \nAnswer:
   ```

   and one token was generated.
3. The hidden state from the last layer of the model was saved off for the generated token. This was then combined with the correctness labels from step (1) to create a dataset of (hidden states, correctness labels).
4. A 3 layer MLP was trained on the dataset of the previous step. This is known in the literature as a probe.
5. The logits of the output of this MLP on held-out validation datasets were converted into probabilities, and the ECE was computed.
6. Temperature scaling was applied here to minimize the ECE, resulting in test dataset ECE of 0.02.

The above follows the procedure of Shen et al. (2024a) for freeform responses, and was applied to both the multiple choice and freeform data for consistency.

Having a calibrated probe model, we then created a teacher dataset, where all datasets were processed by the probe model and the computed probabilities were recorded and quantized in steps of 10% (05% to 95%). This teacher dataset served as the training data for the aLoRA model, which was trained to use the invocation sequence

```
<|start_of_role|>certainty<|end_of_role|>
```

and to generate the quantized percentage values.

B Training datasets for intrinsics

B.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

B.2 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

B.3 Training data for Query Rewrite Intrinsic

The training data contains both: 1) standalone examples, which teach the adapter to refrain from rewriting user questions that are already standalone, and 2) non-standalone examples containing a diversity of patterns that are used to teach the adapter to expand the user turn so that it becomes standalone.

The training data uses the publicly available Cloud corpus of technical documentation pages from MT-RAG.[2] Based on this corpus of documents, we constructed a dataset consisting of high-quality, human-created conversations, where the last turn of the conversation comes into versions: non-standalone version, and corresponding standalone version. The training dataset is proprietary and was obtained in combination with a third-party company who contracted the human annotators.

B.4 Training data for Answerability Determination Intrinsic

The training data uses the publicly available Government corpus from MT-RAG Katsis et al. (2025) as the source of documents. Based on this corpus, we constructed a dataset consisting of a mix of human-created and synthetically generated multi-turn conversations. It includes two types of examples: (1) Answerable queries, where the final user question can be answered based on the provided documents. These examples teach the adapter to recognize when sufficient information is present to support an answer. (2) Unanswerable queries, where the documents lack the necessary information to answer the final user query. We used Mixtral as an automatic judge to validate the answerability labels and filter out noisy samples.

[2] https://github.com/IBM/mt-rag-benchmark