

A Invocation sequences

In our implementation, the aLoRA weights are activated one token after the *start* of the invocation sequence. For the benchmark dataset experiments, we simply set the invocation sequence to be the “generation prompt” specified by the base model’s chat template. For the various intrinsic experiments, we most often used a 3-4 token length sequence which included a one-word description of the task.

The invocation sequence has several 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 an (optional) short prompt to aid the learning process.
- Give the adapter weights a few more tokens to process the input prior to generating the output, often improving performance in practice.

In our current implementation, the invocation sequence denoting the starting point of adaptation is fixed, with the option to have a variable prompt follow it prior to actual generation. In principle, there is no need for any consistency of input sequences so long as the point at which the adapter weights should be turned on can be indicated by some means.

B LLM Transformer architecture

A generic architecture is shown in Figure 9.

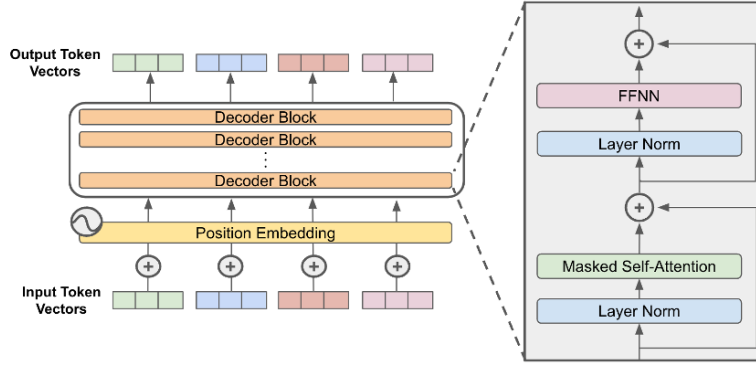


Figure 9: A generic LLM architecture.

C Proofs

We repeat the propositions here from the main text and prove them.

Proposition 3 (Proposition 1 (KV equivalence and aLoRA inference)). *For the causal decoder-only transformers we consider, the keys and values (actually, all internal states) prior to t_{invoke} are identical for the base model and any aLoRA adapter model using (6). Specifically, $K_{1:t_{\text{invoke}}-1}^{\text{base}} = K_{1:t_{\text{invoke}}-1}^{\text{adapter}}$, and $V_{1:t_{\text{invoke}}-1}^{\text{base}} = V_{1:t_{\text{invoke}}-1}^{\text{adapter}}$. Inference with the aLoRA adapted model can be done causally one token at a time (by simply increasing t in (6) iteratively) with KV cache reuse.*

Proof of Proposition 1 (KV equivalence and aLoRA inference). In the proof, for notational simplicity we focus on adapters to the attention blocks, but the treatment for adapters to the MLP blocks is the same. By construction (6), for every token index $i < t_{\text{invoke}}$ the adapter’s weight matrices coincide with the base model’s:

$$W_{\text{adapter}}^Q|_i = W^Q, \quad W_{\text{adapter}}^K|_i = W^K, \quad W_{\text{adapter}}^V|_i = W^V.$$

Hence for each $i < t_{\text{invoke}}$ and starting with the first attention layer,

$$Q_i^{\text{adapter}} = x_i(W^Q + 0) = x_iW^Q = Q_i^{\text{base}},$$

and similarly

$$K_i^{\text{adapter}} = x_iW^K = K_i^{\text{base}}, \quad V_i^{\text{adapter}} = x_iW^V = V_i^{\text{base}}.$$

Since the transformer’s subsequent layer outputs (including all MLP and layer-norm states) are deterministic functions of these Q, K, V up to i , it follows by induction on layer depth that *all* internal states for tokens $1, \dots, t_{\text{invoke}} - 1$ agree between the two models. In particular the cached key- and value-matrices satisfy

$$K_{1:t_{\text{invoke}}-1}^{\text{adapter}} = K_{1:t_{\text{invoke}}-1}^{\text{base}}, \quad V_{1:t_{\text{invoke}}-1}^{\text{adapter}} = V_{1:t_{\text{invoke}}-1}^{\text{base}}.$$

During generation at time $t \geq t_{\text{invoke}}$, both models proceed token-by-token via exactly the same causal-attention mechanism [1], merely appending the new (Q_t, K_t, V_t) row and reusing the previously cached rows. Since up to $t - 1$ those rows coincide, the adapter may *reuse* the base model’s KV cache for the first $t_{\text{invoke}} - 1$ tokens, and then continue to grow its own cache for tokens $t_{\text{invoke}}, \dots, t - 1$. This establishes that aLoRA inference can indeed be performed causally, one token at a time, with full KV cache reuse. \square

Proposition 4 (Proposition 2 (aLoRA vs. LoRA inference costs)). *Consider invoking an adapter with T_{cache} tokens of input for which a base model KV cache exists and $T_{\text{new}} \ll T_{\text{cache}}$ input tokens without cache.⁷ The first token generated by the aLoRA adapter requires $O(T_{\text{cache}}T_{\text{new}})$ operations, while LoRA requires $O((T_{\text{cache}} + T_{\text{new}})^2)$. Furthermore, aLoRA must maintain only $O(T_{\text{new}})$ additional KV cache memory, while LoRA requires additional $O(T_{\text{cache}} + T_{\text{new}})$. If N distinct adapters share the first T_{cache} tokens as input, aLoRA costs become $O(NT_{\text{cache}}T_{\text{new}})$ and $O(NT_{\text{new}})$, while LoRA has $O(N(T_{\text{cache}} + T_{\text{new}})^2)$ and $O(N(T_{\text{cache}} + T_{\text{new}}))$.*

Proof of Proposition 2 (aLoRA vs. LoRA inference costs). Let the input be partitioned into T_{cache} tokens whose base-model cache is already available, and T_{new} tokens on which the adapter must act without precomputed cache. We compare the cost of generating the first new token:

- **aLoRA.** To generate token $t = T_{\text{cache}} + T_{\text{new}} + 1$, we do a forward pass of the transformer, where we have keys and values for the first T_{cache} tokens (and therefore do not need to recompute these). The MLP blocks do not interact between tokens, so scale linearly only with $T_{\text{new}} + 1$, hidden dimension, and number of layers. For each attention layer, we
 1. compute $T_{\text{new}} + 1$ new rows each of Q, K, V in $O(T_{\text{new}}d_k)$,
 2. form the attention scores by multiplying these new queries against the cached-plus-new key-matrix of size $(T_{\text{cache}} + T_{\text{new}}) \times d_k$, costing $O((T_{\text{cache}} + T_{\text{new}})d_k)$,
 3. apply the $T_{\text{new}} + 1$ masked softmaxes and weighted sums over $(T_{\text{cache}} + T_{\text{new}})$ values in $O(T_{\text{new}}(T_{\text{cache}} + T_{\text{new}})d_v)$,
 4. form the adapter’s own new cache entries, yielding at most $O(T_{\text{new}})$ extra storage.

Hence, considering hidden dimension and number of layers as constant, the dominant cost is $O(T_{\text{cache}} + T_{\text{new}}(T_{\text{new}} + T_{\text{cache}})) = O(T_{\text{new}}T_{\text{cache}})$, and extra memory $O(T_{\text{new}})$.

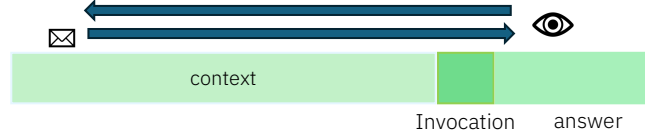
- **LoRA.** Since LoRA’s rank- r updates apply to *all* tokens—including those in the original context—no prefill cache may be reused. Generating the first token thus requires recomputing attention over $(T_{\text{cache}} + T_{\text{new}})$ tokens, incurring $O((T_{\text{cache}} + T_{\text{new}})^2) = O(T_{\text{cache}}^2) \gg O(T_{\text{new}}T_{\text{cache}})$ compute, and storing $O(T_{\text{cache}} + T_{\text{new}})$ key/value rows.

If N distinct adapters each process the same $T_{\text{cache}} + T_{\text{new}}$ context but disjoint T_{new} segments, the aLoRA per-adapter cost scales as $O(T_{\text{cache}} + NT_{\text{new}}T_{\text{cache}})$ time and $O(NT_{\text{new}})$ memory, whereas LoRA’s cost scales on the full quadratic context for each adapter (totaling $O(N(T_{\text{cache}} + T_{\text{new}})^2)$ time and $O(N(T_{\text{cache}} + T_{\text{new}}))$ memory). \square

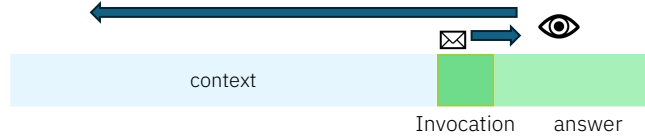
⁷We presume that T_{new} is the invocation sequence and following.

D Adapter Model Capacity and Increased Rank

Figure 10 illustrates this observation (described in the main text). 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.



(a) LoRA is able to modify the keys (K) of task-relevant values (V) in the context attention layers for the modified queries (Q) recover via the attention mechanism.



(b) For the context, aLoRA is not able to modify context KVs, it must rely on learning modified Qs alone to extract information from existing context KVs.

Figure 10: Intuition for why aLoRA adapters often require increased rank r . ALoRA is not able to modify the keys and values of input tokens prior to the invocation sequence to send new “messages” forward to the generation step, and a too small rank constraint limits its capacity to do this.

E Inference Timing Performance evaluation methodology details

In this section we document the methodology followed in the experiments leading to Figure 3. These experiments are modeled, particularly for the smaller prompt lengths (10k tokens) after multi-turn RAG situations where a collection of documents concatenated with multiple conversation turns are followed by an answer of total length a few hundred tokens, and then where judges are used to determine various qualities of the answer.

For each model considered, 5 low rank adapters with random weights were created for ranks 8 and 32, and independent vLLM instances were launched for each model and collection of adapters in two modalities: a server where the adapters are regarded as regular LoRAs (where we used rank 8) and a server where they are regarded as activated LoRAs (where we used rank 32). We note that while in our experiments such high rank discrepancies are the exception rather than the rule, we adopted this setup so as to give an additional potential advantage to LoRA in the evaluation, as a lower rank update is in principle cheaper to compute; we also remark that we saw very little change in the experiments compared to a setting where both ranks match. For each model, the KV cache size in tokens as calculated by vLLM is used to compute the batch size to use by dividing the KV cache size by the length of the sequence that the LLM will be tested on (including initial prompt, generation and evaluation). A total of three batches are tested consecutively for each combination of model, number of evaluators and prompt length. Each batch is processed in sequential stages: answer generation for the entire batch is then followed by one or more evaluations, also done for the entire batch.

Batches are created by choosing prompts at random while ensuring that their length, after tokenization using any given model’s tokenizer, is precisely the desired length. The length of the answers (256 tokens) and evaluations (16 tokens) are enforced by passing to vLLM’s generate method matching minimum and maximum number of tokens to generate.

F Benchmark SFT experiment details

F.1 Models

Llama 3.2 1B Instruct (meta-llama/Llama-3.2-1B-Instruct on Huggingface), Llama 3.2 3B Instruct (meta-llama/Llama-3.2-3B-Instruct), Llama3.1 8B (meta-llama/Llama-3.1-8B-Instruct) [8], and Mistral 7B (mistralai/Mistral-7B-Instruct-v0.3) [31].

F.2 Tasks and Datasets

Here we provide additional information on the SFT tasks in Figure 4. Table 1 gives information for each dataset on the size of the train/validation/test splits, as well as the number of multiple-choice responses for the multiple-choice tasks. Freeform tasks have unstructured natural language strings as target output.

Table 2 provides the Huggingface path for each dataset. The URL can be recovered as <https://huggingface.co/datasets/Lots-of-LoRAs/PATH> where PATH is the name indicated in Table 2.

We next provide the task definition prompt for each of the datasets. Note that the actual datasets include this task definition, in-context-learning (ICL) examples, and the test input as part of the full prompt to the LLM. Examples can be seen on the Huggingface pages for each dataset.

Bengali Hate Speech Classification “In this task, you are given a hateful post in Bengali that expresses hate or encourages violence towards a person or a group based on the protected characteristics such as race, religion, sex, and sexual orientation. You are expected to classify the post into four classes: Religious, Political, Geopolitical and Personal depending on the topic.”

WIQA: Effect Classification “In this task you will be given a process, and a question. The process contains a sequence of steps that happen in order. The question asks about the effect of a certain event on another event. If the first event has a positive effect on the second event, answer with "for", if it has a negative effect, answer with "against". If there’s no causal relationship between the two, answer with "none".”

MMLU Conceptual Physics MCQA “You are given a question on conceptual physics. You are also given 4 answer options (associated with "A", "B", "C", "D"), out of which only one is correct. You need to answer the question by selecting the correct option. You should only answer with the choice letter, not the whole answer.”

MMLU College Computer Science MCQA “You are given a question on college computer science. You are also given 4 answer options (associated with "A", "B", "C", "D"), out of which only one is correct. You need to answer the question by selecting the correct option. You should only answer with the choice letter, not the whole answer.”

SocialIQA Question Generation “In this task, you’re given context and an answer. Your task is to generate the question for this answer based on the given context with commonsense reasoning about social situations.”

Hindi Sentence Perturbation “Given a sentence in Hindi, generate a new Hindi sentence by performing small changes on the sentence. Here, make sure that the changes are semantically related and syntactically similar to the input. And the generated sentence should have high commonsense plausibility, that is to have reasonable probability of it being true.”

SuperGLUE Question Generation “In this task, you are given Wikipedia articles on a range of topics, we ask you to write a question based on the content of the articles that can be answered in a binary manner i.e. True or False.”

Type	Task	#Options	Train	Valid	Test
Multiple Choice	Bengali Hate Speech Classification	3	18.1k	227	227
	WIQA: Effect Classification	3	5.2k	650	650
	MMLU Conceptual Physics MCQA	4	142	18	18
	MMLU College Computer Science MCQA	4	90	12	11
Freeform	SocialIQA Question Generation		5.2k	650	650
	Hindi Sentence Perturbation		5.18k	648	647
	SuperGLUE Question Generation		1.51k	190	189

Table 1: Dataset sizes and option counts for selected tasks.

Task	Name on Lots-of-LoRAs (Huggingface)
Bengali Hate Speech Classification	task1494_bengali_hate_speech_classification
WIQA: Effect Classification	task1727_wiqa_what_is_the_effect
MMLU Conceptual Physics	task693_mmmlu_answer_generation_conceptual_physics
MMLU College Computer Science	task688_mmmlu_answer_generation_college_computer_science
SocialIQA Question Generation	task581_socialiqa_question_generation
Hindi Sentence Perturbation	task407_mickey_hi_sentence_perturbation_generation
SuperGLUE Question Generation	task1660_super_glue_question_generation

Table 2: Task types and their Hugging Face dataset paths on Lots-of-LoRAs.

F.3 Training and evaluation details

Following typical LoRA SFT best practices, for both LoRA and aLoRA we used 4 training epochs, alpha of 32, dropout of 0.05, adapted the K, Q, and V modules in all layers, and searched over ranks [6, 8, 16, 32] and learning rates $[3 \times 10^{-6}, 10^{-5}, 3 \times 10^{-5}, 10^{-4}, 3 \times 10^{-4}]$. Batch size of 8 was used, with 16-bit arithmetic precision. Hyperparameters were selected by taking the configuration that performed best on the validation set, and reported performance was computed on the test set for those selected models. All training runs were done on single H100 GPUs.

All models were evaluated by comparing the generated answers to golden answers provided in the dataset⁸ and percent correctness was used as the metric. For the multiple-choice tasks, agreement was simple to evaluate. For the freeform tasks, an LLM judge prompt compared the generated answers to the golden answers and output a binary decision.

G Additional details for intrinsics experiments

For the intrinsics tasks, all attention weights (keys, queries, values) were adapted in all layers, using rank 32 adapters. The learning rate and number of epochs were tuned to achieve the best validation performance (as was the case for the LoRA adapters of [5]). Intrinsics training tasks each used an 8 GPU H100 node.

G.1 Uncertainty Quantification

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. **Training pipeline** First, a probe-based model was trained to produce calibrated certainty scores, using a large diverse collection of QA datasets detailed in Appendix H.1. Note that throughout, the chat template was used. The procedure for this was as follows:

⁸Note that some datasets may contain some label noise.

- 1025 1. A “meta-dataset” was created containing User inputs, Answer generations (from the base
1026 model), and correctness labels for those generations.
- 1027 2. For each row in the meta-dataset, the base model was prompted with input of the form (User
1028 inputs, Answer generations, meta prompt), where the meta prompt was
1029 `Is the above answer correct?\n <A> Yes, \n No, \nAnswer:`
1030 and one token was generated.
- 1031 3. The hidden state from the last layer of the model was saved off for the generated token. This
1032 was then combined with the correctness labels from step (1) to create a dataset of (hidden
1033 states, correctness labels).
- 1034 4. A 3 layer MLP was trained on the dataset of the previous step. This is known in the literature
1035 as a probe.
- 1036 5. The logits of the output of this MLP on held-out validation datasets were converted into
1037 probabilities, and the ECE was computed.
- 1038 6. Temperature scaling was applied here to minimize the ECE, resulting in test dataset ECE of
1039 0.02.

1040 The above follows the procedure of [28] for freeform responses, and was applied to both the multiple
1041 choice and freeform data for consistency.

1042 Having a calibrated probe model, we then created a teacher dataset, where all datasets were processed
1043 by the probe model and the computed probabilities were recorded and quantized in steps of 10%
1044 (05% to 95%). This teacher dataset served as the training data for the aLoRA model, which was
1045 trained to use the invocation sequence

1046 `<|start_of_role|>certainty<|end_of_role|>`

1047 and to generate the quantized percentage values.

1048 G.2 Answerability determination

1049 The input to the model is a list of conversational turns and a list of documents converted to a string
1050 using `apply_chat_template` function. These turns can alternate between the user and assistant
1051 roles. The last turn is from the user. The list of documents is a dictionary with text field, which contains
1052 the text of the corresponding document. To prompt the aLoRA adapter to determine answerability, a
1053 special answerability role is used to trigger this capability of the model. The role includes the keyword
1054 "answerability": `<|start_of_role|>answerability<|end_of_role|>` When
1055 prompted with the above input, the model generates the answerable or unanswerable output. See [5]
1056 for more details.

1057 **Training Details** The aLoRA and LoRA adapters were fine-tuned under the following regime: rank
1058 = 32, learning rate = 5e-6, number of epochs = 25, with early stopping based on validation set, and
1059 90/10 split between training and validation.

1060 G.3 Query Rewrite

1061 **Usage** The input to the model is a list of conversational turns converted to a string using
1062 `apply_chat_template` function. These turns can alternate between the user and assistant
1063 roles, and the last turn is assumed to be from the user. To prompt the aLoRA adapter to rewrite the
1064 last user turn, a special rewrite role is used to trigger the rewrite capability of the model. The role
1065 includes the keyword “rewrite” followed by a short description of the query rewrite task. Even though
1066 one main application for query rewrite is in RAG settings, this intrinsic can be used to rewrite user
1067 questions for other conversational use cases (e.g., to access a database, or other APIs, or tools). As
1068 such, the adapter does not need any RAG documents (that may be present in the context, in a RAG
1069 setting) and uses only the dialog turns with what is being said between the user and assistant.

1070 **Training** Both the aLoRA and LoRA adapters were fine-tuned under the following regime: rank
1071 = 32, number of epochs = 25, with early stopping based on validation set, and 90/10 split between
1072 training and validation.

1073 **Evaluation data** We evaluate on three different subsets of MT-RAG: a) full MT-RAG dataset (842
1074 data points with last user turns); b) the non-standalone subset of MT-RAG dataset, which is a subset
1075 of 260 (out of 842) last user turns that were annotated by humans as non-standalone (i.e., they
1076 are dependent on the prior context); c) the standalone subset of MT-RAG dataset, which is the
1077 complementary subset, with all the last user turns that were annotated by humans as standalone.

1078 **Answer generation quality** We also evaluate answer generation quality, with top-k passages retrieved
1079 under the various query rewrite strategies for the retriever. We choose here $k = 20$, but similar
1080 trends take place for other values of k . We used Granite-3.2-8b instruct as the answer generator, and
1081 RAGAS Faithfulness (RAGAS-F) and RAD-Bench score as metrics for answer quality. We use the
1082 same three testsets as above.

1083 G.4 Jailbreak Detection

1084 **Usage** The input to the model is a single prompt to assess for harmful content. Cur-
1085 rently, the intrinsics operate on single turn queries. To prompt the aLoRA/LoRA adapters
1086 `<|start_of_role|>jailbreak<|end_of_role|>` is used.

1087 **Training** Both the aLoRA and LoRA adapters were fine-tuned under the following regime: rank =
1088 32, fixed 5,000 optimization steps, $6e-5$ learning rate with the Adam optimizer.

1089 H Training datasets for intrinsics

1090 H.1 QA datasets for Uncertainty Quantification Intrinsic

1091 The following datasets were used for calibration and/or finetuning.

- 1092 • [BigBench](#)
- 1093 • [MRQA](#)
- 1094 • [newsqa](#)
- 1095 • [trivia_qa](#)
- 1096 • [search_qa](#)
- 1097 • [openbookqa](#)
- 1098 • [web_questions](#)
- 1099 • [smiles-qa](#)
- 1100 • [orca-math](#)
- 1101 • [ARC-Easy](#)
- 1102 • [commonsense_qa](#)
- 1103 • [social_i_qa](#)
- 1104 • [super_glue](#)
- 1105 • [figqa](#)
- 1106 • [riddle_sense](#)
- 1107 • [ag_news](#)
- 1108 • [medmcqa](#)
- 1109 • [dream](#)
- 1110 • [codah](#)
- 1111 • [piqa](#)

1112 H.2 Training data for Query Rewrite Intrinsic

1113 The training data contains both: 1) standalone examples, which teach the adapter to refrain from
1114 rewriting user questions that are already standalone, and 2) non-standalone examples containing a
1115 diversity of patterns that are used to teach the adapter to expand the user turn so that it becomes
1116 standalone.

1117 The training data used the publicly available Cloud corpus of technical documentation pages from
1118 MT-RAG.⁹ Based on this corpus of documents, a dataset was created consisting of high-quality,
1119 human-created conversations, where the last turn of the conversation comes into versions: non-
1120 standalone version, and corresponding standalone version.

1121 H.3 Training data for Answerability Determination Intrinsic

1122 The training data uses the publicly available Government corpus from MT-RAG [13] as the source of
1123 documents. Based on this corpus, the dataset consists of a mix of human-created and synthetically
1124 generated multi-turn conversations. It includes two types of examples: (1) Answerable queries, where
1125 the final user question can be answered based on the provided documents. These examples teach the
1126 adapter to recognize when sufficient information is present to support an answer. (2) Unanswerable
1127 queries, where the documents lack the necessary information to answer the final user query. Mixtral
1128 was used as an automatic judge to validate the answerability labels and filter out noisy samples.

1129 H.4 Training data for Jailbreak Intrinsic

1130 The Jailbreak Intrinsic was trained on 40,000 harmful and benign samples. This is composed of
1131 several sub-sampled open source datasets (Gandalf Ignore Instructions [23], Awesome ChatGPT
1132 Prompts [1], BoolQ [3], SAP [6], UltraChat [7], super natural instructions [38]) as well as non-public
1133 prompt datasets of harmful/benign content.

⁹<https://github.com/IBM/mt-rag-benchmark>