AuLoRA: An Automatic, Label-Free Framework for KV Cache Sharing in Multi-LoRA Systems

Anonymous authors
Paper under double-blind review

Abstract

In modern multi-agent and multi-expert systems, KV heterogeneity across models prevents a successor from inheriting its predecessor's KV cache: every handoff re-prefills the entire prefix context, inflating time-to-first-token (TTFT). This paper targets KV cache sharing in multi-LoRA systems, where one base model serves many domain-specific LoRA experts—a memory-efficient multi-expert setup. Here the base model is the natural sender: its KV cache is adapter-agnostic and can be reused by every expert. Existing training-free sharing methods, however, target either a pair of fine-tuned models or agents collaborating on a single task, leaving the base-to-adapter direction largely unexamined. Reusing the base model's KV cache directly leads to substantial performance degradation. We decompose the difference between the base model's and the expert's KV caches into a weight term and a hidden-state term, which correspond to the two existing families of training-free remedies, a low-rank correction and partial recomputation. Experiments show that the two terms are nearly orthogonal, and the hidden-state term dominates, so an exact low-rank correction removes only 1–3% of the KV difference. Recomputation is thus unavoidable in multi-LoRA systems. State-of-the-art recomputation methods rely on offline calibration: they search over consecutive-layer combinations for the one with the best task accuracy on a labeled calibration set, at a cost that grows quadratically with model depth. We present AuLoRA, a training-free KV cache sharing framework for multi-LoRA systems with two components. (i) Two-stage profiling: a label-free algorithm that reduces the search cost complexity from $O(L^2)$ to $O(L)$, where L is the model depth, without per-candidate decoding and with few calibration samples; its profiling result matches the current state of the art, with about $4\times$ faster end-to-end profiling on question answering and over an order of magnitude faster on summarization. (ii) Tail compensation: a LoRA-only mechanism that lets each expert prefill the tail of its own question on top of the shared context KV cache. With negligible compute overhead, it improves quality at a fixed recomputation budget and lowers the budget required for a target quality. Overall, with roughly one third of the layers recomputed, the two components recover 53–95% of the quality lost to direct reuse and reduce the expert's prefill latency by 2.5–$2.7\times$.

1 Introduction

As LLMs are adopted more widely, a single LLM no longer suffices for diverse, complex tasks. However, in modern multi-agent (Hong et al., 2024; Wu et al., 2024a) and multi-expert systems, there is a non-negligible latency issue when a handoff occurs. After the predecessor model finishes decoding, the successor must re-prefill the entire context already processed by the predecessor, owing to a key-value (KV) cache mismatch across models that differ in prompts, architectures, or weights. Reading the same content twice inflates time-to-first-token (TTFT), especially for long contexts. The increasing use of such systems has made KV cache sharing an active research question (Liu et al., 2026; Jeon et al., 2026).

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

Existing methods formulate the problem as a sender-receiver pair, in which the sender passes its KV cache to the receiver. Because direct reuse degrades quality, the goal is to minimize the degradation while reducing TTFT.

These methods fall into three levels: prompt-level, architecture-level, and weight-level. Prompt-level methods focus on a sender and a receiver that share weights and differ only in their context, and repair the cache entries that the new context invalidates (Yang et al., 2025; Yao et al., 2025; Ye et al., 2025; Geng et al., 2026). Architecture-level methods focus on a sender and a receiver from different families or sizes, and train an explicit mapper to translate the KV cache between the two models (Heo et al., 2026; Fu et al., 2026). Weight-level methods, where our work lies, focus on a sender and a receiver that are fine-tuned variants of the same model, and reduce the KV difference either through specific training paradigms (Woo et al., 2026a;b; Greenewald et al., 2025; Ma et al., 2026) or a training-free correction (Liu et al., 2026; Jeon et al., 2026).

A multi-LoRA system, in which many task-specific adapters (Hu et al., 2022) share one copy of a base model and a request is routed to one of them (Wu et al., 2024b; Huang et al., 2024), is a memory-efficient multi-expert setup. In our setting the base model is the sender and prefills every request, and each LoRA expert is the receiver and decodes over the base model's KV cache (Appendix A). The base model is the natural sender here: its cache is computed without any adapter, so every expert can reuse the same copy. However, the KV cache of the base model is not naturally compatible with that of the LoRA experts.

Existing methods in the base-to-adapter setting avoid the mismatch by applying specific training paradigms. ICaRus (Woo et al., 2026a) trains experts to decode over a cache produced by a frozen base model. PrefillShare (Woo et al., 2026b) fine-tunes task models for a shared prefill. Activated LoRA (Greenewald et al., 2025) applies the adapter only to tokens after an invocation point. SCD (Ma et al., 2026) trains per-pair maps that translate the base cache for the expert, and predicts the hidden states of the few layers it patches. However, all of these methods require a highly specialized training paradigm, a requirement that limits the plug-and-play nature of LoRA. We instead target a training-free setting, in which every LoRA expert is fine-tuned independently in a mainstream LoRA configuration.

Recent training-free KV cache sharing methods take two routes: recomputing a contiguous block of layers (Liu et al., 2026), and adding a low-rank correction to the layers that are reused (Jeon et al., 2026). The two routes correspond to the two factors of the KV difference, a weight term and a hidden-state term (§3). LRAgent (Jeon et al., 2026) argues that recomputation can be skipped, and TTFT reduced accordingly, whenever the two caches are similar enough; our experiments show that in the base-to-adapter setting the measured similarity stays below that condition on every expert, though several come close.

The emerging question is therefore whether recomputation can be skipped in the base-to-adapter setting. Our experiments show the two terms to be nearly orthogonal, with the hidden-state term dominating, which limits any exact low-rank correction to a small fraction of the KV difference (§3). Recomputation is therefore unavoidable in multi-LoRA systems.

Recomputation, however, raises the question of which layers to recompute, and state-of-the-art methods answer it with a labeled search whose profiling complexity is $O(L^2)$ (Liu et al., 2026). Reusing one profile across experts keeps the profiling cost down. Our experiments show, however, that the selected block depends on the dataset and differs from expert to expert, and that profiling each expert separately is expensive on long-output tasks (§5.2).

We present AuLoRA, an automatic, label-free framework for KV cache sharing in multi-LoRA systems, composed of two main components. (i) Two-stage profiling selects a candidate block from an analysis signal and settles the final choice with an output probe. (ii) Tail compensation, a LoRA-only mechanism in which each expert prefills the tail of its own question on top of the shared context KV cache, improves quality at negligible cost and remains compatible with any recomputation budget.

Our contributions are summarized as follows:

- We decompose the difference between the base cache and the expert's own KV cache into a weight term and a hidden-state term, and we determine whether recomputation is needed by measuring the angle between the two terms and their relative scale. Because the two

terms are nearly orthogonal, any exact low-rank correction removes only $1 - \sqrt{1-\rho^2}$ of the difference, where ρ is the weight term's relative size.

- We introduce label-free two-stage profiling, which reduces the profiling complexity from $O(L^2)$ to $O(L)$ and achieves a $4\times$ end-to-end speedup on question answering and over an order of magnitude on summarization (§5.2).
- We add tail compensation to the recomputation route, the one combination no prior method occupies (Table 1): it improves quality at a fixed budget, lowers the budget a target quality requires, and we establish its benefit only for base-to-adapter pairs (§5.3).
- We show that the two components are complementary and can recover 53–95% of the lost quality and reduce the expert's TTFT by 2.5–2.7$\times$ (§5).

2 Problem Setting

A request to a multi-LoRA server splits into a shared prefix of N tokens and a suffix of M tokens. The base model prefills the prefix once and passes the resulting KV cache to the expert that the request is routed to; the suffix carries that expert's own last tokens. Both models have L layers, indexed 0 to $L-1$, and h_ℓ denotes the input hidden state of layer ℓ.

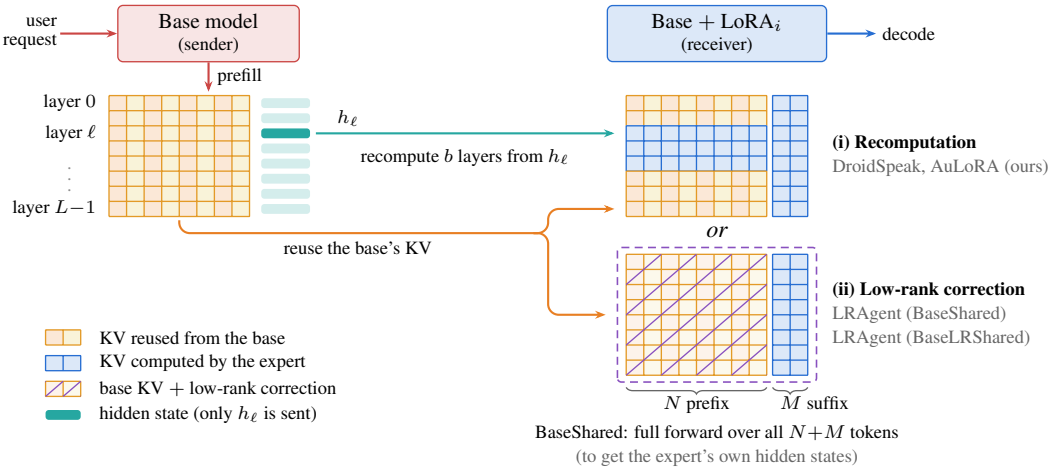


Figure 1: A training-free KV cache sharing method, and what it receives from the sender. Symbols are those of §2; the receiver recomputes b layers starting at layer ℓ, from the hidden state h_ℓ the sender stores there. Orange and yellow cells are the base model's keys and values, reused unchanged; blue cells are recomputed by the expert; the violet hatching in (ii) marks the adapter's low-rank term added on top of a reused cache. Route (i) recomputes a block; route (ii) leaves every layer reused and corrects it instead, and §3 shows why only (i) reaches the term that dominates.

Figure 1 defines the design space of a training-free sharing method. The sender always passes its keys and values, and a method that recomputes also receives the hidden state h_ℓ at one layer; since grouped-query attention (Ainslie et al., 2023) leaves a layer of hidden states larger than a layer of keys and values, by a factor of 2 on Llama and 3.5 on Qwen, only one such layer is ever sent. From there the receiver takes one of two routes. Route (i) recomputes a contiguous block of b layers with its own weights, starting at layer ℓ from the transferred h_ℓ, and reuses the sender's keys and values in the rest. Route (ii) leaves every layer reused and adds the adapter's own low-rank term to the cache instead. One choice is independent of the route: how many of the request's final tokens the receiver computes for itself. With $M = 1$ it computes only the final token, and with M set to the question tail it computes the whole of its own question.

Prefill cost is counted in layer-tokens: pushing one token through one layer costs one layer-token, so a full prefill by the expert costs $L(N+M)$, direct reuse costs LM, and route (i) costs $bN + LM$. Route (ii)'s cost is not apparent from its setting: the BaseShared variant of LRAgent (Jeon et al., 2026) reuses the base keys and values in every layer, so it recomputes nothing, yet it still costs

Table 1: Existing sharing methods, expressed in the design space of Figure 1. N: prefix tokens prefilled by the sender; M: suffix tokens the receiver computes itself; L: layers; b: recomputed layers. Prefill cost is counted in layer-token units. DroidSpeak's sender in fact prefills $N-1$ tokens.

Method	Received from sender	Recomputed layers b	Suffix tokens M	Prefill cost	Selection cost
Full prefill (no sharing)	—	—	—	$L(N+M)$	—
Direct reuse	KV	0	1	LM	—
DroidSpeak	KV + h_ℓ	> 0	1	$bN + LM$	$O(L^2)$, labeled
LRAgent (BaseShared)	KV	0 †	question tail	$L(N+M)$	—
LRAgent (BaseLRShared)	KV	0 ‡	question tail	LM	—
AuLoRA (ours)	KV + h_ℓ	> 0	question tail	$bN + LM$	$O(L)$, **label-free**

† $b = 0$, yet the expert must still run a full forward pass to obtain the hidden states its low-rank correction needs. ‡ BaseLRShared reaches that cost by sharing one frozen low-rank factor across experts, which is built on a specific shared-A training paradigm.

$L(N+M)$, because the low-rank correction it adds needs hidden states that no system stores, leaving the receiver to run all L layers over $N+M$ tokens. What BaseShared saves is memory rather than time. The BaseLRShared variant removes that cost: one frozen low-rank factor shared by every expert makes the correction a single cache, and the receiver prefills only its own tokens.

Table 1 places existing methods in that design space. Throughout the table, b is the recomputation budget, and what a method has to decide for itself is where the recomputed block starts and how many tokens the receiver computes. DroidSpeak (Liu et al., 2026) and AuLoRA take the same route at the same cost, $bN + LM$, and differ only in M and in how they choose ℓ: DroidSpeak searches $O(L^2)$ blocks against labels, whereas AuLoRA reads ℓ off $O(L)$ unlabeled measurements, one per layer.

3 Anatomy of the Sharing Error

The key at layer ℓ is the product of that layer's input and the key projection, $K_\ell = h_\ell W_k$, and an adapter perturbs both factors: it adds a low-rank update AB to the projection, with A the down-projection and B the up-projection, and it shifts the input to $h_\ell + \Delta h_\ell$, because every earlier layer already carries the adapter. Expanding the product splits D_{KV}, the difference between the base cache and the expert's own cache, in two,

$$D_{\mathrm{KV}} \;=\; \underbrace{h_\ell^{\mathrm{base}} AB}_{D_w \text{ (weight term)}} \;+\; \underbrace{\Delta h_\ell \,(W + AB)}_{D_h \text{ (hidden-state term)}}, \tag{1}$$

where W is the key or the value projection, whichever the cache holds. For readability the equation suppresses the norm each projection follows and the adapter's scaling; h_ℓ here is the input the projection sees, and we measure with both in place. We write $\rho = \|D_w\|/\|D_{\mathrm{KV}}\|$ for the weight term's relative size.

Equation 1 separates the two routes of §2, though not symmetrically. What a low-rank correction can add depends on the hidden states it has. From the sender's it is $h_\ell^{\mathrm{base}} AB$, the weight term exactly and nothing else; that is what we measure and what BaseLRShared reaches. BaseShared pays a full forward pass for the expert's own and adds $h_\ell^{\mathrm{expert}} AB$, which differs by $\Delta h_\ell AB$ and recovers no more. Recomputation is the only route that reaches the hidden-state term, and it removes the weight term in the layers it recomputes as well; at the first layer of the block, which is handed the sender's h_ℓ, the two coincide.

What route (ii) is worth is decided by how much of D_{KV} a low-rank correction removes. That depends on two quantities, the angle between the two terms and ρ: if they were parallel, cancelling D_w exactly would shorten D_{KV} by ρ; if they are orthogonal, the residual is $\sqrt{1-\rho^2}$ of the original and the correction shortens the difference by $1 - \sqrt{1-\rho^2}$ (Figure 2a).

Our experiments show that both work against the correction. Across the ten LoRA experts the mean absolute cosine between the two terms is 0.01–0.04 per expert and never exceeds 0.16 at any layer, and ρ is 0.09–0.20 on the value channel, so cancelling D_w exactly shortens D_{KV} by 0.9–3.1%,

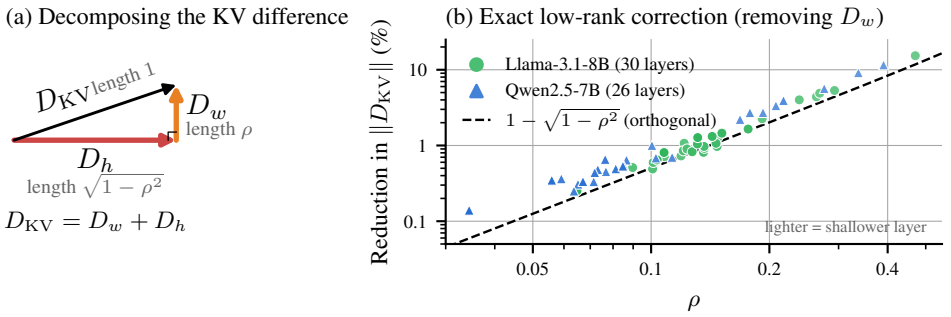


Figure 2: The geometry of the KV difference and the reduction an exact low-rank correction achieves. **(a)** The decomposition of Eq. 1 as vector addition, not to scale. **(b)** The measured reduction in $\|D_{\mathrm{KV}}\|$ against ρ, one point per layer, averaged over the five experts of each family (layers 2–31 of 32 on Llama and 2–27 of 28 on Qwen; value channel; 256 prompts); lighter markers are shallower layers.

against the 0.6–2.7% that the orthogonal expression predicts (Figure 2b). The key channel is flatter still, with ρ of 0.04–0.11 and a shortening of 0.1–0.9%. Because what we add is the adapter's own AB rather than an approximation of it, no correction built from the sender's hidden states can do better than this, and the one that pays for the expert's does not (Appendix B).

LRAgent justifies sharing the base cache by how similar the two caches are, reporting a value cosine of 0.96 between agents that share one frozen A and work on a single task. Read as a condition, that similarity is what allows recomputation to be skipped, so we tested a pair that clears it. Among our ten experts the cosine is 0.87–0.95, just below their figure, whereas a checkpoint only ten optimizer steps from the base model matches at 0.998 on Llama and 0.999 on Qwen. Even at that similarity, an exact low-rank correction recovers almost none of what direct reuse costs there, while recomputation recovers a far larger share of it (Appendix C). What decides the outcome is ρ, and a high cache similarity does not imply a high ρ. Recomputation is thus unavoidable in multi-LoRA systems, and what remains open is how many layers are worth recomputing.

4 AuLoRA

AuLoRA has two components. Two-stage profiling places the recomputed block automatically, from measurements of the base model and the expert, and tail compensation lets each expert prefill the tail of its own question. Neither needs a labeled search, a reference answer, or a change to how the expert was trained. We denote by b the recomputation budget, the number of layers the receiver recomputes, and by ℓ the layer at which the recomputed block starts. Profiling is run once for a pair and returns a block for every budget.

Profiling runs in two stages, as Algorithm 1 states. The first stage measures, for every layer, how far the base model's cache is from the expert's own: at layer ℓ the discrepancy is $r_\ell = \|D^{(\ell)}_{\mathrm{KV}}\| / \|\mathrm{KV}^{\mathrm{expert}}_{(\ell)}\|$, read off one forward pass of each model over the same prompt, on the value channel, where the difference is about 1.7× its counterpart on keys. The anchor is the contiguous window of b layers whose discrepancies sum highest. Figure 3 plots r_ℓ for both families: the profile peaks in the middle of the stack, and at the ten-layer budget the anchor falls at layers 8–17 on Llama and 12–21 on Qwen; a block fixed at the middle of the stack is worse (Appendix D.1).

The anchor and its two neighbours are nearly tied. Moving the window one layer in either direction changes its summed discrepancy by only 0.5–2.9%. That is a tie as far as the first stage can tell, but the blocks are not equivalent: which one the receiver recomputes changes the score the expert reaches. The first stage therefore returns three candidates, $\ell - 1$, ℓ and $\ell + 1$, and hands the choice to the output probe, which reads what the expert produces instead of what its cache contains (Appendix D.2 measures wider windows).

The probe scores a candidate by how far the shared cache moves the expert's own output. The expert first answers each profiling prompt over its own full prefill, and that answer becomes the target:

Algorithm 1 Two-stage label-free profiling, 16 prompts per stage.

Require: base model W, expert $W+AB$, budgets $\mathcal{B}$, unlabeled prompts $\mathcal{P}$ (stage 1) and $\mathcal{Q}$ (stage 2); layers are indexed 0 to $L-1$
Ensure: a recomputation block $[\ell_b^\star, \ell_b^\star+b-1]$ for every $b \in \mathcal{B}$

Stage 1: the layer curve, measured once
for each prompt in $\mathcal{P}$ **do**
  forward the base and the expert; accumulate $\|V_l^{\text{base}}-V_l^{\text{expert}}\|^2$ and $\|V_l^{\text{expert}}\|^2$ per layer
end for
$r_l \leftarrow \sqrt{\sum\|V_l^{\text{base}} - V_l^{\text{expert}}\|^2}\big/\sqrt{\sum\|V_l^{\text{expert}}\|^2}$ for $l = 0, \ldots, L-1$
Stage 2: output probe, reference answers generated once
for each prompt $q \in \mathcal{Q}$ **do**
  $a_q \leftarrow$ greedy answer of the expert with full prefill ▷ one generation per prompt
  $p_q^{\text{ref}} \leftarrow$ expert's next-token distributions, full prefill, teacher-forced on a_q
end for
for $b \in \mathcal{B}$ **do**
  $\ell_0 \leftarrow \arg\max_{0\le\ell\le L-b} \sum_{j=\ell}^{\ell+b-1} r_j$ ▷ $O(L)$ windows of the same curve
  for $\ell \in \{\ell_0-1,\ \ell_0,\ \ell_0+1\}$ **do**
    $p_q^{\ell} \leftarrow$ share with block $[\ell, \ell+b-1]$ and the deployed suffix M, teacher-forced on a_q
    $s_\ell \leftarrow \sum_q \sum_t \mathrm{KL}\big(p_{q,t}^{\text{ref}} \,\|\, p_{q,t}^{\ell}\big) \big/ \sum_q |a_q|$
  end for
  $\ell_b^\star \leftarrow \arg\min_\ell s_\ell$
end for
return $\{\ell_b^\star\}_{b\in\mathcal{B}}$

for each candidate block the probe accumulates, over the tokens of that answer, the KL divergence between the expert's next-token distributions over its own prefill and over the shared cache. Distillation minimizes the same quantity (Hinton et al., 2015); here it ranks configurations rather than trains one. The candidate with the smallest divergence per answer token wins (Appendix D.3). Nothing in the score refers to a reference answer, so the probe needs no labels; each candidate is scored by a single teacher-forced pass over that answer, which yields the divergence at every one of its tokens without decoding it again. The probe runs with the suffix the deployment will use, so that it scores the configuration that will be served.

Profiling cost is a product of three factors: the prompts a protocol needs, the candidate blocks it scores on them, and the cost of scoring one block. A labeled search scores every block on the grid, $O(L^2)$ of them across budgets, and pays a decoded answer for each. AuLoRA scores none in its first stage and three per budget in its second, $O(L)$ across budgets, and pays one teacher-forced pass over an answer generated once per prompt. Every factor is smaller, and the last by the widest margin on long answers, where decoding pays for each token in turn while a teacher-forced pass reads them all at once (Appendix E).

The second component, tail compensation, is a LoRA-only mechanism that changes how the request is divided rather than how the cache is repaired. Both routes of Figure 1 act on entries the receiver has inherited from the base model; the tail removes the inheritance instead. The sender stops its prefill at the end of the shared context, and the receiver computes the tokens of its own question itself, so that at those positions no entry of the cache comes from the base model and every layer is produced by the expert's own weights. The weight term of Equation 1 therefore vanishes at those positions, and what is left of the hidden-state term is only what reaches them through attention over the shared context. The length of the tail is the length of the question rather than a parameter to tune, and because the tail leaves the shared cache untouched, it composes with any budget, including $b = 0$.

5 Experiments

5.1 Setup and main results

We train ten LoRA experts with one recipe and no tuning per task (Appendix F.1): five tasks on each of Llama-3.1-8B-Instruct (Grattafiori et al., 2024) and Qwen2.5-7B-Instruct (Qwen Team et al.,

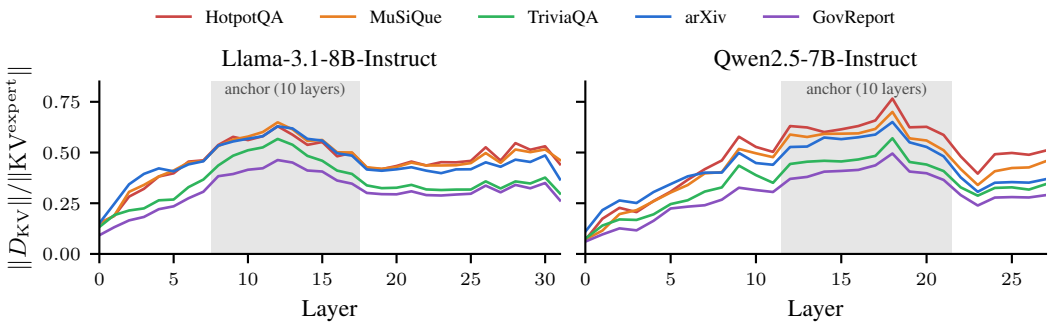


Figure 3: The per-layer discrepancy stage 1 reads, and the anchor it selects. Vertical axis: $\|D_{\mathrm{KV}}\|/\|\mathrm{KV}^{\mathrm{expert}}\|$ on the value channel, 256 prompts. The shaded band is the anchor at a ten-layer budget. The mid-stack peak is an empirical regularity, not a derived property.

2024), namely HotpotQA (Yang et al., 2018), MuSiQue (Trivedi et al., 2022), TriviaQA (Joshi et al., 2017), arXiv (Cohan et al., 2018) and GovReport (Huang et al., 2021), each at 12–16k tokens of context. We report token-level F_1 (Rajpurkar et al., 2016) on the three question-answering tasks and ROUGE-L (Lin, 2004) on the two summarization tasks, over held-out test sets. Every method profiles on the same data: a validation split held out from each expert's training data and disjoint from the test set, which is the kind of data DroidSpeak uses as well. Two further experts on a sixth task (Dasigi et al., 2021) are set aside (Appendix F.2), because direct reuse costs them nothing we can distinguish from zero, which leaves recovery undefined. Recovery throughout is the share of the gap between a full prefill by the expert and direct reuse that a setting closes, computed per expert and then averaged. Intervals are 95% throughout: a paired bootstrap (Koehn, 2004) over questions where two settings are compared on one expert, and a Bayesian bootstrap (Rubin, 1981; Benavoli et al., 2017) over experts where one method is compared with another across the ten. We call two methods indistinguishable when that interval contains zero (Appendix F.3).

Table 2 compares the methods at their published protocols. Direct reuse loses over half of what fine-tuning gained, and neither of the training-free repairs closes much of it: LRAgent's BaseShared recovers part of the gap while spending a full prefill (Appendix B), and DroidSpeak, profiled once per family as its paper prescribes, recovers a little more. AuLoRA recovers 53–95% of the gap: it is better than DroidSpeak on eight of the ten experts and indistinguishable on the other two. It spends about a third of a full prefill and cuts the expert's time to first token by 2.5–2.7× (Appendix F.4). The tail costs about ten percent more time to first token than the same block with a single suffix token, and buys 2.0 points of score averaged over the ten experts; choosing the block rather than taking DroidSpeak's buys a further 1.3 (Table 3). Two of the ten experts start from a gap of only 1.9 and 3.0 points, so their recovery fractions carry wide intervals and should not be read as the top of the range.

Separating the three repairs with the block fixed at the anchor, so that layer selection plays no part, leaves recomputation as the only one that moves the number on every task group (Table A11): without a tail it takes recovery from nothing to 53%, and with one from 42% to 70%. The low-rank correction, computed from the sender's hidden states at every layer, is worth between -2 and $+4$ points of recovery averaged over the ten experts and between -4 and $+6$ within a task group, which is what §3 predicts. The tail helps everywhere but by amounts that depend strongly on the task: on single-hop question answering it alone recovers 75%, because the gap there is only two or three points to begin with, whereas on summarization it recovers 26%, and summarization is where we recover least overall (Appendix G).

5.2 Layer selection

Layer selection can be compared on its own by giving both sides the same budget and a single suffix token, so that only the choice of block differs (Table 3). DroidSpeak's published protocol recovers 45% and returns the same block on only 47% of resampled profiling sets. Giving it its best chance, we strengthened it to profile every expert separately on 128 questions, which raises it to 50% recovered and 74% stability (Appendix H). Only then does it reach us: the paired difference is

Table 2: Comparison with existing sharing methods under a 10-layer recomputation budget, each with its published protocol. LRAgent: BaseShared. DroidSpeak: published protocol; stronger variants in Table 3. Suffixes are those of Table 1. Bold: best among the sharing methods. *: AuLoRA better than DroidSpeak (paired bootstrap over questions, 95% interval above 0). The last two rows are Llama / Qwen. TTFT: median of 30 runs at 16k tokens, one H100, batch 1 (Appendix F.4). Both profiling methods are scored as an expectation over 500 resampled profiling sets.

Task	Model	Full prefill	Direct reuse	LRAgent (BaseShared)	DroidSpeak	AuLoRA (ours)	Recovery (%)
HotpotQA	Llama	82.8	69.4	77.0	76.2	**80.0***	79
	Qwen	79.4	67.0	70.5	72.4	**74.8***	64
MuSiQue	Llama	70.5	42.5	55.4	58.5	**66.7***	87
	Qwen	61.1	34.3	43.7	44.2	**52.6***	68
TriviaQA	Llama	92.0	90.1	91.7	91.8	**91.9**	95
	Qwen	91.5	88.5	90.8	89.8	**91.0***	82
arXiv	Llama	39.8	33.0	34.3	36.3	**36.9**	58
	Qwen	40.7	34.5	36.6	36.2	**37.8***	53
GovReport	Llama	43.8	38.2	39.3	40.3	**41.1***	53
	Qwen	43.4	35.8	37.9	36.7	**40.8***	66
Average, QA / summ.		79.6 / 41.9	65.3 / 35.4	71.5 / 37.0	72.2 / 37.4	**76.2 / 39.2**	79 / 58
Prefill compute (% of full)		100 / 100	≈0	100 / 100	31 / 36	31 / 36	
TTFT at 16k (ms)		1107 / 1024	—	1149 / 1067	363 / 379	405 / 412	

Table 3: Layer selection on its own: the same receiver protocol ($M = 1$), a ten-layer budget, ten experts. Δ *score*: AuLoRA minus the row, in metric points, with a 95% interval (expert-level Bayesian bootstrap). *Profiling cost*: end-to-end GPU-hours per expert on one H100; DroidSpeak is priced from rates measured on an optimized engine and we are measured on our prototype, and Appendix E prices both sides on one engine. Table A13 adds two rows, among them a split-half oracle chosen with the test set's own labels, which we match to within 0.22 points, $[-0.53, +1.26]$, on the same bootstrap as this column. ‡ Profiled once on the HotpotQA expert and reused by the family. § One greedy generation per prompt as the teacher-forcing target; no decoding per candidate.

Method (profiling prompts)	Labels / decoding	Recovery (%)	Δ score (points)	Profiling cost (GPU-h) QA / summ.
DroidSpeak, 50 prompts‡	yes / yes	45	$+1.34$ $[+0.58, +2.36]$	0.05–0.06 / 0.05–0.06
DroidSpeak per expert, 128	yes / yes	50	$+0.82$ $[-0.06, +1.86]$	0.60–1.1 / 13–18
Stage 1 only, 16	no / no	53	$+0.31$ $[-0.16, +1.03]$	0.01–0.02 / 0.01
AuLoRA (ours), 16 + 16	no / no§	**56**	—	0.15–0.29 / 0.10–0.18

$+0.82$ points in our favour with an interval of $[-0.06, +1.86]$, so the two are indistinguishable. The profiling speedup we report is against this strengthened variant: profiling it costs 3.6–4.2× more than ours on question answering and 94–146× more on summarization (estimated; Appendix E). The published protocol is cheaper, at the sacrifice of performance and stability. Stage one alone is perfectly repeatable and still finds the best block only half the time; the probe ends 0.50 points closer to the best block over fifty-two expert-budget cells, with an interval of $[0.21, 0.86]$, for a few minutes per expert (Appendices H and H.1). The profile is not specific to adapters: on the fully fine-tuned pair DroidSpeak itself uses, its labeled search and our stage 1 select indistinguishable blocks (Appendix I).

5.3 Recomputation budget and the tail

Figure 4 traces the same recipe across budgets on the four multi-hop question answering experts. The tail is worth most where the budget is smallest: at two recomputed layers it takes recovery from 3.1% to 47.6%, and a recipe with no recomputation at all but a tail already recovers 42.5%, which a single suffix token does not match until eight or nine layers. Six layers with a tail roughly match ten layers without one, which is the sense in which the tail lowers the budget a target quality requires. The upper line of Figure 4 is an oracle: the best block on DroidSpeak's grid, given the tail as well,

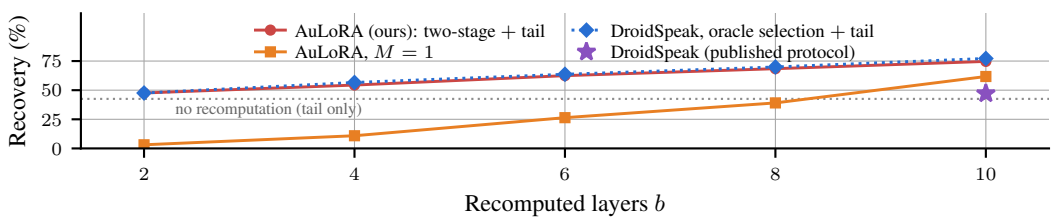


Figure 4: Recovery against the recomputation budget, on the four multi-hop question-answering experts. The star is DroidSpeak's published protocol, measured at ten layers only.

chosen with the labels of the test set itself. It bounds from above what DroidSpeak can reach with a tail at any budget, and no deployable protocol can see those labels. We stay at or just below it across the whole range, with no labels at all. We report quality up to ten layers, the budget of Table 2; half of a full prefill is reached at sixteen layers on Llama and fourteen on Qwen, and a receiver that spends that much is competing with recomputing the context outright.

6 Related Work

LRAgent (Jeon et al., 2026) has two variants, and only BaseLRShared escapes the full forward pass; it escapes it by sharing one frozen A, the down-projection of the adapter's update, across agents. That sharing has to be built into training, which rules out adapters fine-tuned independently of one another, so the variant that is cheap is also the one our setting cannot use. DroidSpeak (Liu et al., 2026) carries no such condition, and its labeled search is what §5.2 compares against.

The methods of §1 that change training remove the same difference at training time instead of at inference. Among them SCD (Ma et al., 2026) is the one that also has to choose layers: it translates most layers through learned per-pair maps, patches a few by predicting their inputs, and picks those few by restoring one layer at a time and scoring a task metric. Its maps and patch predictors cost about ten GPU-hours and 1.6 gigabytes of auxiliary weights for one pair of 7B models, where our profile costs minutes and adds no weights at all.

Selecting layers without labels is not by itself new. KVComm (Shi et al., 2026) transfers a subset of layers and drops the context in the rest, and picks that subset from attention mass and a mid-depth prior with a single calibration prompt; what it selects, however, are layers to keep rather than layers to recompute, so it repairs nothing and sits orthogonal to both routes above. Its receiver prefills its own new tokens, as LRAgent's does, but as a property of the setting rather than a mechanism either one measures.

7 Limitations and Conclusion

Every expert in our experiments carries the adapter on all projections at rank 32, a mainstream configuration among publicly released adapters (Appendix J) but not the only one in use. Restricting the adapter to fewer projections moves a larger share of the difference into the weight term, where a low-rank correction can reach it, and we have not measured how far that shifts the balance.

AuLoRA is, to our knowledge, the first training-free recipe that shares a base model's KV cache with LoRA experts fine-tuned independently of it: an analysis that shows why recomputation cannot be skipped, a two-stage profile that locates the block to recompute from $O(L)$ unlabeled measurements rather than an $O(L^2)$ labeled search and at a fraction of its profiling cost, and a tail whose gain we establish only for base-to-adapter pairs (Appendix I). Together they recover 53–95% of what direct reuse costs on about a third of the layers, and cut the expert's time to first token by 2.5–2.7×.

AI USE STATEMENT

We used generative AI tools in writing this paper and in producing its analysis code. Drafts of the prose were produced with an AI assistant and then rewritten and corrected by the authors, who take responsibility for every sentence that remains. The scripts that read the experiment logs and produce the tables and figures were also written with AI assistance; each number reported here was recomputed from the raw result files, and the scripts are released with the paper so that the computation can be checked. AI assistance was used to search for and compile the bibliography, and every entry was verified against the publisher's own record. We also used an AI assistant as a discussion partner on how to present the results: which evidence supports which claim, and how the sections are ordered. The research question, the experimental design, the measurements themselves and the claims we draw from them are the authors' own, and no other AI system was consulted.

ETHICS STATEMENT

This work studies inference efficiency and involves no human subjects and no new data collection. The base models and the datasets are public and are used under their own licences. The method trades quality for latency, and the paper reports the loss it causes rather than only the speedup, so that an operator can see what a given budget costs. Cache sharing sometimes raises a privacy concern, because a cache computed for one request may carry content from it; in our setting the cache is shared between two models handling the same request, never between requests or between users, so no content crosses that boundary.

REPRODUCIBILITY STATEMENT

The setting and its notation are fixed in §2, the method is stated as pseudocode in Algorithm 1, and §5 gives the experimental protocol. The fine-tuning recipe is in Appendix F.1, the latency and profiling measurement conditions in Appendix F.4, and the sampling procedure behind the adapter survey in Appendix J. Every method in the comparison draws its profiling prompts from the same pool, a held-out slice of each expert's training split that is disjoint from the test set. We release the adapter weights, the profiles, the per-question scores behind every table, and the analysis scripts that turn them into the numbers reported here.

REFERENCES

Joshua Ainslie, James Lee-Thorp, Michiel de Jong, Yury Zemlyanskiy, Federico Lebron, and Sumit Sanghai. GQA: Training generalized multi-query transformer models from multi-head checkpoints. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 4895–4901, Singapore, December 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.emnlp-main.298. URL https://aclanthology.org/2023.emnlp-main.298/.

Alessio Benavoli, Giorgio Corani, Janez Demšar, and Marco Zaffalon. Time for a change: a tutorial for comparing multiple classifiers through Bayesian analysis. *Journal of Machine Learning Research*, 18(77):1–36, 2017. URL http://jmlr.org/papers/v18/16-305.html.

Arman Cohan, Franck Dernoncourt, Doo Soon Kim, Trung Bui, Seokhwan Kim, Walter Chang, and Nazli Goharian. A discourse-aware attention model for abstractive summarization of long documents. In Marilyn Walker, Heng Ji, and Amanda Stent (eds.), *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 2 (Short Papers)*, pp. 615–621, New Orleans, Louisiana, June 2018. Association for Computational Linguistics. doi: 10.18653/v1/N18-2097. URL https://aclanthology.org/N18-2097/.

Pradeep Dasigi, Kyle Lo, Iz Beltagy, Arman Cohan, Noah A. Smith, and Matt Gardner. A dataset of information-seeking questions and answers anchored in research papers. In Kristina Toutanova, Anna Rumshisky, Luke Zettlemoyer, Dilek Hakkani-Tur, Iz Beltagy, Steven Bethard, Ryan Cotterell, Tanmoy Chakraborty, and Yichao Zhou (eds.), *Proceedings of the 2021 Conference of*

the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, pp. 4599–4610, Online, June 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.naacl-main.365. URL https://aclanthology.org/2021.naacl-main.365/.

Tianyu Fu, Zihan Min, Hanling Zhang, Jichao Yan, Guohao Dai, Wanli Ouyang, and Yu Wang. Cache-to-cache: Direct semantic communication between large language models. In *The Fourteenth International Conference on Learning Representations (ICLR)*, pp. 43130–43158, 2026. URL https://proceedings.iclr.cc/paper_files/paper/2026/file/474ada926b331d78f06d95e8913111cc-Paper-Conference.pdf.

Yingsheng Geng, Yuchong Gao, Weihong Wu, Guyue Liu, and Jiang Liu. RelayCaching: Accelerating LLM collaboration via decoding KV cache reuse. In *Proceedings of the Forty-Third International Conference on Machine Learning (ICML)*, 2026. URL https://icml.cc/virtual/2026/poster/66638. To appear. PMLR proceedings not yet published as of 2026-09-26. arXiv:2603.13289.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, et al. The llama 3 herd of models, 2024. URL https://arxiv.org/abs/2407.21783.

Kristjan Greenewald, Luis Lastras, Thomas Parnell, Vraj Shah, Lucian Popa, Giulio Zizzo, Chulaka Gunasekara, Ambrish Rawat, and David Cox. Activated LoRA: Fine-tuned LLMs for intrinsics. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 38, pp. 53022–53049. Curran Associates, Inc., 2025. doi: 10.52202/085713-1769. URL https://proceedings.neurips.cc/paper_files/paper/2025/file/4d0b6303d4a4811445f69f357bf6def5-Paper-Conference.pdf.

Taekyung Heo, Rasoul Shafipour, Ritchie Zhao, Maximilian Golub, Mohammad Mahdi Kamani, Ritika Borkar, Makesh Tarun Chandran, Pantea Zardoshti, and Bita Darvish Rouhani. Cross-model KV cache transfer in LLM families: A closed-form linear mapping for prefill reuse, 2026. URL https://arxiv.org/abs/2608.03893. arXiv:2608.03893. No formal publication found as of 2026-09-26.

Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network, 2015. URL https://arxiv.org/abs/1503.02531. NIPS 2014 Deep Learning Workshop.

Sirui Hong, Mingchen Zhuge, Jiaqi Chen, Xiawu Zheng, Yuheng Cheng, Ceyao Zhang, et al. MetaGPT: Meta programming for a multi-agent collaborative framework. In *The Twelfth International Conference on Learning Representations (ICLR)*, 2024. URL https://openreview.net/forum?id=VtmBAGCN7o.

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. In *The Tenth International Conference on Learning Representations (ICLR)*, 2022. URL https://openreview.net/forum?id=nZeVKeeFYf9.

Chengsong Huang, Qian Liu, Bill Yuchen Lin, Tianyu Pang, Chao Du, and Min Lin. LoraHub: Efficient cross-task generalization via dynamic LoRA composition. In *First Conference on Language Modeling (COLM)*, 2024. URL https://arxiv.org/abs/2307.13269.

Luyang Huang, Shuyang Cao, Nikolaus Parulian, Heng Ji, and Lu Wang. Efficient attentions for long document summarization. In Kristina Toutanova, Anna Rumshisky, Luke Zettlemoyer, Dilek Hakkani-Tur, Iz Beltagy, Steven Bethard, Ryan Cotterell, Tanmoy Chakraborty, and Yichao Zhou (eds.), *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pp. 1419–1436, Online, June 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.naacl-main.112. URL https://aclanthology.org/2021.naacl-main.112/.

Hyesung Jeon, Hyeongju Ha, and Jae-Joon Kim. LRAgent: Efficient KV cache sharing for multi-LoRA LLM agents. In *Proceedings of the Forty-Third International Conference on Machine Learning (ICML)*, 2026. URL https://icml.cc/virtual/2026/poster/61572. To appear. PMLR proceedings not yet published as of 2026-09-26. arXiv:2602.01053.

Mandar Joshi, Eunsol Choi, Daniel Weld, and Luke Zettlemoyer. TriviaQA: A large scale distantly supervised challenge dataset for reading comprehension. In Regina Barzilay and Min-Yen Kan (eds.), *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 1601–1611, Vancouver, Canada, July 2017. Association for Computational Linguistics. doi: 10.18653/v1/P17-1147. URL https://aclanthology.org/P17-1147/.

Philipp Koehn. Statistical significance tests for machine translation evaluation. In *Proceedings of the 2004 Conference on Empirical Methods in Natural Language Processing*, pp. 388–395, Barcelona, Spain, July 2004. Association for Computational Linguistics. URL https://aclanthology.org/W04-3250/.

Chin-Yew Lin. ROUGE: A package for automatic evaluation of summaries. In *Text Summarization Branches Out*, pp. 74–81, Barcelona, Spain, July 2004. Association for Computational Linguistics. URL https://aclanthology.org/W04-1013/.

Yuhan Liu, Yuyang Huang, Jiayi Yao, Shaoting Feng, Zhuohan Gu, Kuntai Du, Hanchen Li, Yihua Cheng, Junchen Jiang, Shan Lu, Madan Musuvathi, and Esha Choukse. DroidSpeak: KV cache sharing across fine-tuned model variants. In *23rd USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, pp. 319–338, Renton, WA, USA, May 2026. USENIX Association. ISBN 978-1-939133-54-0. URL https://www.usenix.org/conference/nsdi26/presentation/liu-yuhan.

Qianli Ma, Zhiqing Tang, Hanshuai Cui, Zhi Yao, and Weijia Jia. Semantic cache distillation: Efficient state transfer via reuse and selective patching. In *Proceedings of the Forty-Third International Conference on Machine Learning (ICML)*, 2026. URL https://icml.cc/virtual/2026/poster/63533. To appear. PMLR proceedings not yet published as of 2026-09-26. arXiv:2606.07684.

Qwen Team, An Yang, et al. Qwen2.5 technical report, 2024. URL https://arxiv.org/abs/2412.15115.

Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. SQuAD: 100,000+ questions for machine comprehension of text. In Jian Su, Kevin Duh, and Xavier Carreras (eds.), *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pp. 2383–2392, Austin, Texas, November 2016. Association for Computational Linguistics. doi: 10.18653/v1/D16-1264. URL https://aclanthology.org/D16-1264/.

Donald B. Rubin. The Bayesian bootstrap. *The Annals of Statistics*, 9(1):130–134, 1981. doi: 10.1214/aos/1176345338.

Xiangyu Shi, Marco Chiesa, Gerald Q. Maguire Jr., and Dejan Kostic. KVComm: Enabling efficient LLM communication through selective KV sharing. In *The Fourteenth International Conference on Learning Representations (ICLR)*, pp. 84546–84573, 2026. URL https://proceedings.iclr.cc/paper_files/paper/2026/file/883105b282fe15275991b411e6b200c5-Paper-Conference.pdf.

Harsh Trivedi, Niranjan Balasubramanian, Tushar Khot, and Ashish Sabharwal. MuSiQue: Multihop questions via single-hop question composition. *Transactions of the Association for Computational Linguistics*, 10:539–554, 2022. doi: 10.1162/tacl_a_00475. URL https://aclanthology.org/2022.tacl-1.31/.

Sunghyeon Woo, Jaeeun Kil, Hoseung Kim, Minsub Kim, Joonghoon Kim, Ahreum Seo, Sungjae Lee, Minjung Jo, Jiwon Ryu, Baeseong Park, Se Jung Kwon, and Dongsoo Lee. ICaRus: Identical cache reuse for efficient multi-model inference. In *The Fourteenth International Conference on Learning Representations (ICLR)*, pp. 32937–32957, 2026a. URL https://proceedings.iclr.cc/paper_files/paper/2026/file/37f6be32f832caf0f7980469fb06165b-Paper-Conference.pdf.

Sunghyeon Woo, Hoseung Kim, Sunghwan Shim, Minjung Jo, Hyunjoon Jeong, Jeongtae Lee, Joonghoon Kim, Sungjae Lee, Baeseong Park, Se Jung Kwon, and Dongsoo Lee. PrefillShare: A shared prefill module for KV reuse in multi-LLM disaggregated serving, 2026b. URL https:

//arxiv.org/abs/2602.12029. arXiv:2602.12029. No formal publication found as of 2026-09-26.

Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Beibin Li, Erkang Zhu, et al. AutoGen: Enabling next-gen LLM applications via multi-agent conversation. In *First Conference on Language Modeling (COLM)*, 2024a. URL https://openreview.net/forum?id=BAakY1hNKS.

Xun Wu, Shaohan Huang, and Furu Wei. Mixture of LoRA experts. In *The Twelfth International Conference on Learning Representations (ICLR)*, 2024b. URL https://proceedings.iclr.cc/paper_files/paper/2024/hash/ce806d8b4bf38cd92d483e5a0490d983-Abstract-Conference.html.

Jingbo Yang, Bairu Hou, Wei Wei, Yujia Bao, and Shiyu Chang. KVLink: Accelerating large language models via efficient KV cache reuse. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 38, pp. 133797–133824. Curran Associates, Inc., 2025. doi: 10.52202/085713-4461. URL https://proceedings.neurips.cc/paper_files/paper/2025/file/c217d123885341853cecbdd2be809983-Paper-Conference.pdf.

Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William Cohen, Ruslan Salakhutdinov, and Christopher D. Manning. HotpotQA: A dataset for diverse, explainable multi-hop question answering. In Ellen Riloff, David Chiang, Julia Hockenmaier, and Jun'ichi Tsujii (eds.), *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pp. 2369–2380, Brussels, Belgium, October-November 2018. Association for Computational Linguistics. doi: 10.18653/v1/D18-1259. URL https://aclanthology.org/D18-1259/.

Jiayi Yao, Hanchen Li, Yuhan Liu, Siddhant Ray, Yihua Cheng, Qizheng Zhang, Kuntai Du, Shan Lu, and Junchen Jiang. CacheBlend: Fast large language model serving for RAG with cached knowledge fusion. In *Proceedings of the Twentieth European Conference on Computer Systems (EuroSys)*, pp. 94–109, Rotterdam, Netherlands, 2025. ACM. doi: 10.1145/3689031.3696098.

Hancheng Ye, Zhengqi Gao, Mingyuan Ma, Qinsi Wang, Yuzhe Fu, Ming-Yu Chung, Yueqian Lin, Zhijian Liu, Jianyi Zhang, Danyang Zhuo, and Yiran Chen. KVCOMM: Online cross-context KV-cache communication for efficient LLM-based multi-agent systems. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 38, pp. 17882–17928. Curran Associates, Inc., 2025. doi: 10.52202/085713-0605. URL https://proceedings.neurips.cc/paper_files/paper/2025/file/1a074a28c3a6f2056562d00649ae6416-Paper-Conference.pdf.

A THE MULTI-LORA SETTING

Figure A1 sketches the arrangement this paper assumes. One base model serves several LoRA experts, each fine-tuned independently of the base model and of the others. A request carries a long context; the base model prefills that context once, and the expert the request is routed to reuses the resulting cache instead of prefilling the context for itself. §2 states the same arrangement in the notation the rest of the paper uses.

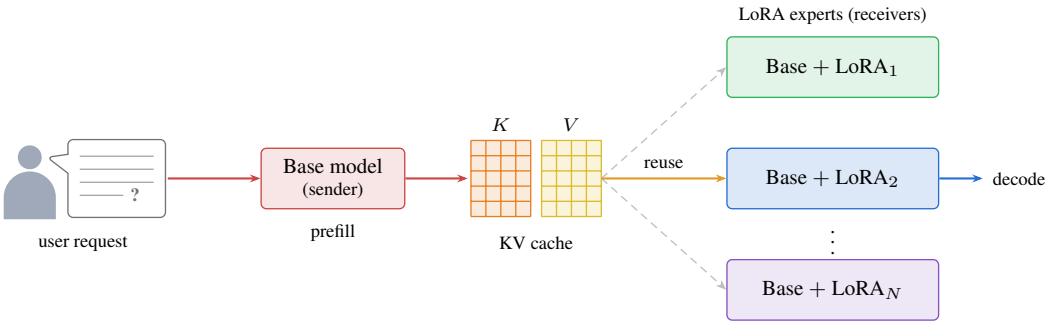


Figure A1: A multi-LoRA system. The base model prefills each long-context request once, and the expert that the request is routed to reuses the resulting KV cache and decodes from it.

B THE LRAGENT BASELINE, PER EXPERT

Table A1 reports what BaseShared costs and what it recovers. The variant reuses the base keys and values everywhere and adds each expert's low-rank term, and LRAgent does not say whether that term is added to keys as well as values. We implemented both readings. On the six experts where both were run they recover 54.6% and 53.6% of the gap on average, so the choice does not change the conclusion. What the variant costs is the whole point: the layer-tokens it must prefill are 99.99% of a full prefill, and in our measurements a receiver using it takes 1.03 to 1.09 times as long as prefilling the context itself. With a single suffix token it recovers 2.6% of the gap on average, between -2.8% and 9.8%. With the question tail it recovers 42.9%, but a tail on its own, at no recomputation and no forward pass, already recovers 42.4%: over the ten experts the extra full prefill buys 0.5 points of recovery.

Table A1: What BaseShared costs and what it recovers, over the ten experts. The correction needs hidden states the sender does not store, so the receiver runs every layer over $N+M$ tokens whichever suffix it uses; the layer-tokens it must prefill are 99.99% of a full prefill and a receiver using it takes 1.03 to 1.09 times as long as prefilling the context itself. Adding the low-rank term to keys as well as values rather than to values alone recovers 54.6% against 53.6% on the six experts where we ran both, so that reading does not change the conclusion.

Setting	Receiver's forward pass	Recovery (%)
BaseShared, single suffix token	every layer, $N+M$ tokens	2.6
BaseShared, question tail	every layer, $N+M$ tokens	42.9
Tail only, no correction	M tokens	42.4

C A PAIR AS CLOSE AS THE PREMISE REQUIRES

§3 argues that cache similarity does not decide how much a low-rank correction recovers. To put that on a pair far closer than any of our experts, we saved intermediate checkpoints of one adapter and evaluated the one ten optimizer steps from the base model, whose value cache matches the base at a cosine of 0.998 on Llama and 0.999 on Qwen. Table A2 reports each setting against a full prefill by that checkpoint, on 2000 questions.

Table A2: Score relative to a full prefill by the expert on an adapter ten optimizer steps from the base model, HotpotQA, 2000 questions, 95% paired bootstrap intervals. Zero is the receiver prefilling for itself; every other row is below it, so closer to zero is better. The low-rank correction, which adds $h_\ell^{\text{base}} AB$ at every layer, and BaseShared, which pays a full forward pass to use the expert's own hidden states instead, stay within 0.14 points of direct reuse even at this similarity, whereas recomputation does not.

Setting	Llama	Qwen
Receiver prefills for itself	0	0
Direct reuse	-2.00 $[-3.25, -0.80]$	-2.71 $[-3.84, -1.62]$
Low-rank correction, every layer	-1.86 $[-3.08, -0.66]$	-2.57 $[-3.68, -1.50]$
BaseShared	-1.87 $[-3.08, -0.70]$	-2.67 $[-3.79, -1.59]$
Ten recomputed layers	-1.33 $[-2.35, -0.33]$	-0.70 $[-1.51, +0.10]$
Ten recomputed layers + tail	-0.51 $[-1.13, +0.10]$	-0.36 $[-0.93, +0.25]$

Across the checkpoints at 10, 25, 50, 100 and 125 steps and the final adapter, evaluated on 500 questions, the value cosine falls from 0.998 to 0.884 on Llama and from 0.999 to 0.866 on Qwen, while ρ stays between 0.046 and 0.147; the checkpoint at 25 steps, at 0.950 and 0.970, is the one that matches the figure LRAgent reports. What direct reuse costs tracks what fine-tuning gained rather than the distance in steps, at 41–59% of the gain on Llama and 29–58% on Qwen: the Qwen checkpoint at 25 steps gained more than the one at 50 and lost more with it, 5.78 points against 3.12.

D Design choices in the profile

Three choices in the profile are not forced by the analysis, and each has a control.

D.1 A fixed mid-stack block

The profile of Figure 3 peaks mid-stack, which invites the question whether stage 1 is an expensive way of arriving at a mid-depth prior. We compared the anchor against a block fixed at the middle of the stack and against three blocks drawn at random, on three pairs — the two HotpotQA experts of the main text and the fully fine-tuned pair of Appendix I — at two budgets each, with both sides computing a single suffix token, on 500 questions and with 95% paired bootstrap intervals.

Table A3 reports every comparison. The anchor is better in twenty-two of the twenty-four, by 2.7 to 15.2 metric points; against the mid-stack block on its own it is better in five of the six, by 2.7 to 10.9. The two blocks differ by only three layers on the main-text pairs, so the margin does not come from the mid-stack block being far away.

Table A3: The anchor against a block fixed at the middle of the stack and against three blocks drawn at random, at two budgets on each of three pairs. *Anchor*: the score the anchor block reaches. Each remaining column is the anchor minus that control, in metric points, with a 95% paired bootstrap interval over 500 questions; both sides compute a single suffix token. Two intervals include zero, and both are visible here: the mid-stack block at eight layers on Llama and the third random block on the fully fine-tuned pair at ten layers.

Pair	b	Anchor	Mid-stack	Random 1	Random 2	Random 3
Llama HotpotQA	8	76.0	$+1.56$ $[-0.47, +3.68]$	$+3.52$ $[+1.75, +5.49]$	$+6.42$ $[+4.02, +9.16]$	$+5.60$ $[+3.16, +8.09]$
Llama HotpotQA	10	77.9	$+2.74$ $[+0.96, +4.78]$	$+9.08$ $[+6.32, +12.01]$	$+8.83$ $[+6.06, +11.73]$	$+6.03$ $[+3.57, +8.75]$
Qwen HotpotQA	10	73.2	$+4.13$ $[+1.68, +6.61]$	$+4.57$ $[+1.99, +7.10]$	$+3.90$ $[+1.27, +6.53]$	$+5.32$ $[+2.69, +8.03]$
Qwen HotpotQA	14	76.1	$+4.27$ $[+1.95, +6.45]$	$+7.89$ $[+4.85, +10.99]$	$+4.90$ $[+2.59, +7.17]$	$+4.68$ $[+1.90, +7.46]$
Fully fine-tuned pair	10	62.6	$+8.19$ $[+5.01, +11.52]$	$+7.19$ $[+4.07, +10.34]$	$+7.72$ $[+4.74, +10.92]$	-1.50 $[-4.28, +1.26]$
Fully fine-tuned pair	16	69.4	$+10.93$ $[+7.69, +14.62]$	$+15.01$ $[+11.22, +18.89]$	$+15.01$ $[+11.22, +18.89]$	$+15.17$ $[+11.58, +19.01]$

D.2 Why three candidates

Sixteen prompts are enough for stage 1, in that they select the anchor that 256 prompts select on all ten experts. Drawing on the full bank of profiling prompts instead of sixteen moves the recovery from 53–95% to 50–94%. Stage 2 compares the anchor with its two one-layer shifts, and the width

of that window is a design choice we measured. Table A4 reports it. These are the fifty-two configurations of Appendix H. Having a second stage at all is what matters: the anchor on its own lands on a best block in 55.8% of them and falls 0.99 points short, and one layer either side raises that to 84.6% and 0.48 with three candidates. Widening further buys little: two and three layers reach 86.3% and 88.2% for five and seven candidates, and the worst single configuration is 2.35 points from one layer outward whatever the width.

Table A4: How wide the second stage's window should be. *Candidates*: blocks the probe scores at each budget. *Best block found*: the share of 52 expert-budget configurations in which the rule lands on a block that ties with the best one measured; the two widest rules are scored on 51, because one configuration has no candidate three layers out. *Shortfall*: metric points below the best block, averaged over the configurations, and for the single worst one.

Rule	Candidates	Best block found (%)	Mean shortfall	Worst shortfall
Stage 1 anchor only	1	55.8	0.99	4.48
One layer either side (ours)	3	84.6	0.48	2.35
Two layers either side	5	86.3	0.41	2.35
Three layers either side	7	88.2	0.38	2.35

D.3 Two signals on the same candidates

A cheaper method could still be the wrong method, so we held the candidate blocks and the profiling questions fixed and changed only the signal that ranks them. The blocks are the three the baseline's two-layer grid reaches around the anchor rather than the anchor and its one-layer shifts, because per-question accuracy exists only on that grid; both signals see the same three. Table A5 reports both. Scoring the candidates by task accuracy on the profiling set, which is what DroidSpeak's search does, lands on a best block for five of the ten experts; scoring them by the divergence the probe measures lands on one for seven. The largest single difference is Llama MuSiQue, where the accuracy signal chose a block 6.36 points below the best available one while the probe chose that block.

Table A5: The same candidate blocks ranked by two signals. Both see the same three blocks, the anchor and the two shifts that DroidSpeak's two-layer grid reaches, which is where per-question accuracy exists; these are not the one-layer shifts AuLoRA scores. Both use their own full bank of profiling questions: 128 to 312 for the accuracy signal, 128 to 256 for the probe. *Best block*: the highest-scoring of the three on the test set. *Short*: metric points below it. *Ties* counts a block whose paired difference from the best has an interval containing zero and a point estimate within one metric point: the accuracy signal ties on five of the ten experts and the probe on seven.

		Accuracy on the profiling set		Output probe	
Expert	Best block	Block	Short	Block	Short
Llama HotpotQA	8–17	8–17	0.00	8–17	0.00
Qwen HotpotQA	13–22	14–23	1.77	14–23	1.77
Llama MuSiQue	8–17	6–15	6.36	8–17	0.00
Qwen MuSiQue	13–22	14–23	2.18	14–23	2.18
Llama TriviaQA	9–18	8–17	0.07	8–17	0.07
Qwen TriviaQA	11–20	14–23	0.85	12–21	0.17
Llama arXiv	7–16	8–17	0.13	8–17	0.13
Qwen arXiv	9–18	12–21	1.28	10–19	0.69
Llama GovReport	5–14	6–15	0.73	8–17	2.35
Qwen GovReport	10–19	10–19	0.00	10–19	0.00

E The profiling cost, in full

Profiling cost decomposes into three factors: the prompts a protocol needs for a stable answer, the candidate blocks it scores on them, and what scoring one candidate costs. Table A6 gives all three, and the totals they produce.

Table A6: Profiling cost by its three factors, and the totals they lead to. The factors do not multiply to the totals: AuLoRA also pays stage 1 and one generation per prompt whatever the budget, blocks at other budgets cost in proportion to their layer count, and the per-block row is measured on each side's own stack while the totals are priced as Appendix E describes. *Blocks scored*: across every even budget from two layers upwards, on a 32-layer and a 28-layer model. *Scoring one block*: at the ten-layer budget, on the Llama HotpotQA expert, whose answers are a few tokens, and the Llama GovReport expert, whose answers run to hundreds. *Totals*: end-to-end GPU-hours per expert on one H100, over the ten experts. The first two factors differ by about the same amount on both task types; the third is what separates them, and it is the one AuLoRA changes in kind rather than in degree.

	Labeled search (strengthened, per expert)	**AuLoRA (ours)**
Prompts	128	**16**
Blocks scored, Llama / Qwen	136 / 105	**46 / 40**
Scoring one block, question answering	0.57 s, one generation	**0.51 s, one forced pass**
Scoring one block, summarization	14.94 s, one generation	**0.29 s, one forced pass**
Paid once per pair	—	two forwards, one answer per prompt
Total, question answering (GPU-h)	0.60–1.1	**0.15–0.29**
Total, summarization (GPU-h)	13–18	**0.10–0.18**

The first two factors are protocol constants and behave the same on both task types. A labeled search scores the whole grid, $(L - b)/2 + 1$ blocks at budget b and $O(L^2)$ across budgets, and its strengthened form profiles every expert on 128 questions. AuLoRA scores three candidates per budget, $O(L)$ across budgets, on 16; it also pays, once for a pair and whatever budgets follow, the two forward passes of stage 1 and one greedy answer per prompt, which the totals include.

The third factor is what separates the task types, because a labeled search decodes an answer for every candidate while we read a teacher-forced pass over an answer that already exists. Where answers are a few tokens the two cost almost the same; where they run to hundreds they differ by a factor of fifty. Our own cost is measured on our prototype; the labeled search is estimated from its measured prefill and per-token decode rates, with one sequence in flight, and the estimate scales with the number of output tokens, which is four on the question answering experts and 512 on the summarization experts. Hence the resulting ratio, 3.6–4.2× on question answering and 94–146× on summarization.

The third factor is also a count of generated tokens, and the ratio of the two counts does not depend on how that generation is scheduled: a labeled search decodes an answer for each of the 136 candidate blocks on Llama, or 105 on Qwen, on each of its 128 prompts, while we decode one answer on each of 16 prompts, between 840 and 1,088 times fewer generated tokens. Batching lowers what a decoded token costs on both sides; it does not change that count. Repricing both sides on the same engine, at the largest batch the GPU holds, raises the question answering ratio to 14–16× and lowers the summarization ratio to 33–63×; the numbers we report put us on the slower engine and the baseline at one sequence in flight.

F Experimental details

F.1 Training recipe

Every expert is fine-tuned with the same recipe, and no hyperparameter search was run: the values below were fixed before the first expert was trained and never revisited. Table A7 gives it. We keep the checkpoint with the best downstream score on a development set of 500 examples carved out of the training split, which is disjoint from the test set and is also the pool every profiling method draws from.

The three question answering tasks are padded to length. HotpotQA and MuSiQue ship contexts of about 1.4k tokens, so we lengthen each one with paragraphs borrowed from other questions of the same split until it reaches 12 or 16k. The borrowed paragraphs are shuffled together with the original ones under a per-example seed rather than appended, so the gold paragraphs do not sit at the front; any borrowed paragraph containing the answer is skipped; and the pool is drawn from the

Table A7: The recipe every expert is trained with. No hyperparameter search was run: these values were fixed before the first expert was trained and never revisited.

Target modules	q, k, v, o and the three MLP projections
Rank / α / dropout	32 / 64 / none
Bias, rank-stabilized scaling	none
Optimizer	AdamW in bfloat16
Learning rate	2×10^{-4}, cosine schedule, 3% warmup
Weight decay	0.01
Effective batch	32 sequences
Annealing	one cycle; two for the long-context experts
Sequence length	truncated to 14,336 tokens from the middle, as in the LongBench evaluation script
Schedule	a fixed 1,000 steps, one checkpoint every 125, no early stopping
Seed	3407 for the data shuffle, the held-out split and training
Training pool	the training split shuffled and capped at 40,000 examples, of which a 1,000-step run at an effective batch of 32 consumes 32,000
Checkpoint	the one with the best downstream score on 500 examples held out of the training split, disjoint from the test set

training split, disjoint from what we evaluate on. TriviaQA is natively long and is used as it ships, as are the two summarization tasks.

Two hygiene checks are worth reporting. Splitting MuSiQue by row leaks 96.8% of its development questions, because single-hop sub-questions recur across compositions; we split by connected component instead, which brings that to zero, and the residual on HotpotQA is 2.7%. The padding pool was shared between the training and evaluation halves in an earlier run, which would let a model learn to recognize and skip filler; it is disjoint in the runs reported here. Borrowed paragraphs are topically unrelated to the question, which may make them easier to ignore than the retrieved distractors a benchmark ships with, so our padding is, if anything, the easier kind.

F.2 The sixth task

Ten of the twelve experts we trained appear in the main text; Table A8 reports the two we set aside. They are the two trained on the sixth task, and we set them aside by a rule fixed in advance: an expert enters the main tables only if the quality a full prefill holds over direct reuse is distinguishable from zero. On these two it is not, at $+1.90$ points with a 95% interval of $[-0.39, +4.08]$ and $+1.14$ with $[-0.63, +2.84]$; on the other ten the interval excludes zero everywhere, the narrowest being $+1.93$ with $[+0.94, +2.94]$. A recovery percentage divides by that gap, and dividing by a quantity consistent with zero produces intervals such as $[-215, +112]\%$, which carry no information.

The exclusion says nothing about how well the adapters learned the task. Fine-tuning lifts these two experts by $+9.87$ points with $[+7.38, +12.36]$ and $+8.76$ with $[+6.05, +11.35]$, from 42.53 to 52.40 and from 43.71 to 52.48, both comfortably significant. What is small is the loss that sharing causes, not the gain that fine-tuning produced: the loss is 19% and 13% of the gain here, against a median of 57% across the ten experts in the main text.

Table A8: Why the two experts on the sixth task are set aside. An expert enters the main tables only if the quality a full prefill holds over direct reuse is distinguishable from zero. Recovery divides by that gap, and dividing by a quantity consistent with zero produces intervals such as $[-215, +112]\%$. The exclusion is about the gap, not about the adapter: fine-tuning lifts both experts by an amount that is comfortably significant.

Expert	Base → fine-tuned	Gain from fine-tuning (points)	Full prefill − direct reuse (points)
Sixth task, first expert	42.53 → 52.40	$+9.87$ $[+7.38, +12.36]$	$+1.90$ $[-0.39, +4.08]$
Sixth task, second expert	43.71 → 52.48	$+8.76$ $[+6.05, +11.35]$	$+1.14$ $[-0.63, +2.84]$
Narrowest of the ten we keep	—	—	$+1.93$ $[+0.94, +2.94]$

F.3 STATISTICS

Table A9 collects the procedures and what they give. Where §3 quotes a range for ρ or for the cosine it is a range across the ten experts, each aggregated over layers; Figure 2b plots one point per layer instead, so its spread is wider. Where two settings are compared on one expert, we resample its questions in pairs, 4000 times, and report the 2.5th and 97.5th percentiles of the paired difference. Where one method is compared with another across the ten experts, we resample the experts with a Bayesian bootstrap, which is the level at which the two methods actually differ. Calling an interval that contains zero "indistinguishable" understates the case slightly, so we also report the probability that the difference lies inside a region of practical equivalence of half a point. Against DroidSpeak's published protocol we win with probability 0.99; against its per-expert variant on fifty questions, 0.89; against the same variant on 128 questions the split is 0.75 win, 0.25 equivalent, 0.00 lose, which is the sense in which the two match.

Two further sources of optimism are worth separating out. A block chosen on the test set and then scored on the same test set is flattered: our difference from it is -0.25 points with an interval of $[-0.52, -0.02]$; estimating it on one half and scoring on the other leaves our difference from that oracle at $+0.19$ with $[-0.21, +0.48]$, and the best block itself changes between halves for up to 61% of the experts. Charging our own profile for the luck of drawing one profiling set, rather than treating its choice as fixed, leaves the point estimate at $+0.80$ and widens the interval from $[-0.07, +1.80]$ to $[-0.40, +2.48]$, so the conclusion survives the most conservative accounting we can write down.

Table A9: The procedure behind every interval, and what it gives against each DroidSpeak variant. The region of practical equivalence is half a point. An interval that contains zero understates the case, which is why the probabilities are reported alongside it.

Comparison	Procedure
Two settings on one expert	paired bootstrap over its questions, 4000 draws, 2.5th and 97.5th percentiles of the paired difference
One method against another, across the ten	Bayesian bootstrap over experts, the level at which the methods differ

AuLoRA against	Win	Equivalent	Lose
DroidSpeak, published protocol	0.99		
per expert, fifty questions	0.89		
per expert, 128 questions	0.75	0.25	0.00

F.4 MEASUREMENT CONDITIONS

Table A10 states the conditions behind both costs we report. Time to first token is the marginal cost of one expert: the base cache and the stored hidden state are already resident on the same H100, the receiver's cache points at the base tensors rather than copying them, and timing runs from the moment the receiver is handed the request to the moment the first token's logits are available. Every figure is the median of thirty runs at a 16k-token context with batch size one, on our HuggingFace prototype and with no transfer between machines.

DroidSpeak itself fixes no budget but takes the smallest block whose drop stays within 5%. Calibration cost covers a full sweep over every even budget, which is what either method needs before a budget-quality trade-off can be drawn, and it uses DroidSpeak's own two-layer granularity. The two sides are not timed on the same software stack: DroidSpeak's search is priced from rates measured on an optimized inference engine, ours is measured on our HuggingFace prototype, so the slower implementation is on our side. Sweeping every budget instead of every second one raises the ratio in favour of our profiling to 7.1–$8.3\times$ on question answering and 243–$348\times$ on summarization under the same accounting as Appendix E, so the table reports the conservative version of the budget sweep.

Table A10: The conditions behind the two costs we report.

Quantity	How it is measured
Time to first token	median of thirty runs at a 16k-token context, batch size one, on one H100 and our HuggingFace prototype, with no transfer between machines; the base cache and the stored hidden state are already resident, and the receiver's cache points at the base tensors rather than copying them timed from the moment the receiver is handed the request to the moment the first token's logits are available, so it is the marginal cost of one expert
Profiling cost	a full sweep over every even budget, which is what either method needs before a budget-quality trade-off can be drawn, at DroidSpeak's own two-layer granularity ours measured on the prototype, the labeled search priced from rates measured on an optimized inference engine, so the slower implementation is on our side
DroidSpeak's budget rule	the smallest block whose drop stays within 5%

Table A11: Ablation of recomputation, the low-rank correction and the tail, as recovery (%) of the quality lost to direct reuse (0: direct reuse; 100: a full prefill by the expert), averaged within each task group (multi-hop: four experts; single-hop: two; summarization: four). Recomputation always uses the 10-layer anchor block, so no layer selection is involved. A few single-hop cells exceed 100, because the gap there is only two or three points and sharing occasionally scores above a full prefill. The low-rank arm adds $h_\ell^{\text{base}} AB$ at every layer, which is the correction bounded in §3 and the mechanism BaseLRShared uses.

Recompute 10 layers	Low-rank correction	Multi-hop QA		Single-hop QA		Summarization		All 10 experts	
		$M{=}1$	tail	$M{=}1$	tail	$M{=}1$	tail	$M{=}1$	tail
—	—	0	43	0	75	0	26	0	42
—	✓	4	42	1	80	6	25	4	43
✓	—	57	74	80	91	36	56	53	70
✓	✓	56	73	83	92	36	52	53	68

G Ablation

Table A12 gives the grouped numbers of Table A11 one expert at a time. The pattern behind the averages is visible in every row: without a tail, no repair that leaves the layers alone moves the number at all, and the two experts whose recovery exceeds 100% are the two whose gap is under five points, where the percentage is noisy by construction.

H Layer selection against DroidSpeak

Table A13 compares us against a variant of DroidSpeak that we strengthened, and we strengthened it because its published protocol does not return a stable answer. Resampling the profiling set, we recorded how often a protocol returns the block it returns most often, and how often that block ties with the best block on the test set. The published protocol returns the same block on 47% of resampled sets and ties with the best block on 34% of the experts. Profiling every expert separately on the same fifty questions raises those figures to 53% and 41%, and on 128 questions to 74% and 49%. At 128 the choice is stable enough that a head-to-head measures the selection rule rather than the noise of the profiling sample, which is why Table A13 reports that variant alongside the published one. Our own stage 1 returns the same block on every resampled set and ties with the best block on 50% of the experts; adding the probe lowers repeatability to 78% and raises the tie rate to 84%.

A wider grid makes that difference measurable. Over fifty-two cells—the four multi-hop experts at every even budget from two to ten layers, all ten experts at ten layers, each scored both with a single suffix token and with the tail—the probe lands on the best block in 84.6% of them and the anchor alone in 55.8%, a paired difference of 28.8 percentage points with an interval of $[13.5, 44.2]$ clustered by expert. The probe ends 0.48 points below the best block on average and the anchor

Table A12: The ablation of Table A11, per expert. *Gap* is the quality a full prefill holds over direct reuse, in points; the eight remaining columns are recovery (%) of that gap, for each combination of recomputation and low-rank correction, with a single suffix token ($M{=}1$) or with the question tail. Recomputation always uses the ten-layer anchor, so layer selection plays no part. †Gap below five points, where a recovery percentage is noisy and can exceed 100.

					Reuse		Low-rank		Recompute		Both	
Expert	n	Full	Reuse	Gap	$M{=}1$	tail	$M{=}1$	tail	$M{=}1$	tail	$M{=}1$	tail
llama_hotpotqa_pad12k	500	82.85	69.37	13.48	0	58	4	58	64	80	63	86
qwen7b_hotpotqa_pad12k	500	79.35	66.98	12.37	0	32	8	28	50	64	50	58
llama_musique_pad16k	500	70.50	42.54	27.96	0	45	1	47	76	91	75	91
qwen7b_musique_pad16k	500	61.11	34.28	26.83	0	36	0	34	40	62	38	59
llama_triviaqa_rc12k	2000	91.98	90.05	1.93	0	89	-0	87	101	99	97	105
qwen7b_triviaqa_rc12k	2000	91.52	88.51	3.01	0	61	3	72	59	83	70	79
llama_arxiv	200	39.77	32.99	6.78	0	20	5	23	60	60	50	54
qwen7b_arxiv	200	40.68	34.47	6.20	0	46	4	34	26	54	35	52
llama_gov_report	200	43.81	38.17	5.64	0	18	11	20	32	50	33	43
qwen7b_gov_report	200	43.38	35.84	7.54	0	20	5	24	27	58	24	57

Table A13: Layer selection under the same receiver protocol ($M = 1$), 10-layer budget, 10 experts. Δ *score*: AuLoRA minus the row, in metric points rather than recovery percentage points, with a 95% interval (expert-level Bayesian bootstrap). *Profiling cost*: end-to-end GPU-hours per expert on one H100, covering every even budget (the conservative choice; Appendix F.4). *vs AuLoRA*: that cost divided by AuLoRA's; above 1 means AuLoRA is that many times cheaper. DroidSpeak's search is priced from rates measured on an optimized inference engine and ours is measured on our HuggingFace prototype; Appendix E prices both sides on one engine. ‡ Profiled once on the HotpotQA expert and reused by all experts of the same family. § One greedy generation per prompt as the teacher-forcing target; no decoding per candidate.

Method (profiling prompts)	Labels / decoding	Recovery (%)	Δ score (points)	Profiling cost (GPU-h) QA / summ.	vs AuLoRA ($\times$)
DroidSpeak, 50 prompts‡	yes / yes	45	$+1.34$ $[+0.58, +2.36]$	0.05–0.06 / 0.05–0.06	0.21–0.33 / 0.35–0.51
DroidSpeak per expert, 50	yes / yes	46	$+1.04$ $[+0.17, +2.06]$	0.24–0.43 / 5.0–7.1	1.4–1.6 / 37–57
DroidSpeak per expert, 128	yes / yes	50	$+0.82$ $[-0.06, +1.86]$	0.60–1.1 / 13–18	3.6–4.2 / 94–146
Stage 1 only, 16	no / no	53	$+0.31$ $[-0.16, +1.03]$	0.01–0.02 / 0.01	0.06–0.07 / 0.05–0.06
AuLoRA (ours), 16 + 16	no / no§	56	—	0.15–0.29 / 0.10–0.18	1
Oracle (split half)	test labels	56	$+0.22$ $[-0.53, +1.26]$	—	—

0.99, a difference of 0.50 points with an interval of $[0.21, 0.86]$. A block drawn at random from the same three candidates lands on the best one 57.1% of the time, so what the second stage buys is the choice among them and not the window they lie in.

H.1 How DroidSpeak's calibration behaves

Two objections to §5.2 are worth answering with data. The first is that we gave the baseline too few questions. Table A14 answers it: we profiled it three times on disjoint sets of fifty, and its own rule, which takes the smallest block whose drop stays within five percent, chose 22, 8 and 32 layers on Llama and 12, 14 and 10 on Qwen; our stage 1 returned the same block all three times on both. In four of nine profiling sets the rule degenerated to no recomputation at all, because direct reuse already sat inside its tolerance.

The second is that more questions would close the gap. Profiling every expert separately and raising the count to the limit of our question bank, two experts do converge: one reaches the best block at 360 questions and another at 256. Three others stay at zero from fifty questions to the limit, and one falls from 40% to 0% as questions are added. Averaged over six experts, going from fifty to 256 questions moves the hit rate from 27% to 34% and the shortfall from 2.17 to 1.80 points. What the extra questions cannot fix is a bias in where the signal points.

Table A14: DroidSpeak's own rule, run three times on disjoint sets of fifty questions. The budget it asks for moves by a factor of four on Llama; our stage 1 returned the same block all three times on both models. Raising the count does not fix it: averaged over six experts, going from fifty to 256 questions moves the hit rate from 27% to 34% and the shortfall from 2.17 to 1.80 points.

Profiling set	Llama	Qwen
First	22 layers	12 layers
Second	8 layers	14 layers
Third	32 layers	10 layers
Our stage 1, all three	same block	same block

I A FULLY FINE-TUNED PAIR

Our experts are adapters, and the natural question is whether the profile works on the pair DroidSpeak itself used. We ran it on that pair and that benchmark. Table A15 reports every recipe we ran there. DroidSpeak's own rule asks for 22 of the 32 layers and recovers 95% of the gap; given that same budget, the block our stage 1 selects is behind it by an amount that does not separate from zero, and its block was chosen with labels while ours was not. Tail compensation is the part that does not carry over. A tail is worth having only if it recovers a share of the gap worth having and it adds to a recomputation block; on this pair only the first holds. On its own it recovers 16%, less than the ten-layer block it would be added to, and put on top of that block it changes the score by -2.30 with $[-5.72, +1.25]$.

Table A15: Every recipe we ran on the fully fine-tuned pair DroidSpeak uses, on its own benchmark: LongBench-E HotpotQA, 300 questions, untruncated. *Recovery*: against a 19.71-point gap. The last column is the row minus the block DroidSpeak's labeled search selects, with a 95% paired bootstrap interval.

Setting	Layers	Score	Recovery (%)	Δ vs DroidSpeak's block (points)
Full prefill by the expert	—	63.84	100	$+1.04$ $[-1.64, +3.98]$
DroidSpeak's labeled search	22	62.80	95	—
Our stage 1, same budget	22	60.26	82	-2.54 $[-6.02, +0.80]$
Our anchor	10	47.99	20	-14.81 $[-19.48, -10.01]$
Our anchor + tail	10	45.69	8	-17.10 $[-21.64, -12.27]$
Tail only	0	47.32	16	-15.48 $[-20.24, -10.68]$
Direct reuse	0	44.13	0	-18.67 $[-23.66, -13.72]$

J ADAPTER CONFIGURATIONS ON HUGGINGFACE

The claim in §1 that our experts use a mainstream LoRA configuration rests on a survey of adapter configuration files published on HuggingFace. We collected candidate repositories two ways, by download count and by upload date, read each `adapter_config.json`, kept the entries whose `peft_type` is LoRA, discarded diffusion, vision, speech, encoder and vision-language models, and kept those whose module names follow the convention shared by the Llama, Qwen, Mistral and Gemma families. Because a single author can upload hundreds of adapters, we report each sample both per repository and with one adapter per author.

Table A16: Target modules of published LoRA adapters. Percentages are of the sample in that row; the remaining combinations, seven in the largest sample, account for at most 5.8%.

Sample	n	all projections	q, v	q, k, v, o	q, k, v
Most downloaded, per repository	520	**74.2%**	13.7%	5.4%	1.0%
Most downloaded, per author	338	**74.3%**	12.7%	7.1%	0.6%
Recently uploaded, per repository	842	**87.5%**	3.9%	5.2%	0.0%
Recently uploaded, per author	221	**79.6%**	5.0%	13.1%	0.0%

Table A16 reports the result: adapters that carry every projection are the majority in all four samples, at 74–88%. The other settings we use are mainstream on the same evidence: a scaling ratio $\alpha/r = 2$

covers 69% and 79% of the two repository-level samples, rank 32 is the first or second most common rank (20.6% and 36.0%, against 37.1% and 35.0% for rank 16), and Qwen and Llama are the two most frequent base families in both. What the survey does not establish is that our conclusions transfer to the minority settings; §7 states that limit.