LRCC: GENERALIZING LOW-RANK COMPRESSION WITH CONDITIONAL COMPUTATION

Anonymous authors
Paper under double-blind review

ABSTRACT

Low-rank compression reduces the cost of pretrained language models by replacing linear transformations with low-rank factorizations. However, conventional methods use a fixed rank allocation during inference, assigning the same amount of compute regardless of the input token. We introduce Low-Rank Conditional Computation (LRCC), which adds token-dependent computation to pretrained models by training one lightweight router per Transformer block to select among a small set of nested low-rank paths. During training, the low-rank factors remain frozen, and only the routers are optimized. We evaluate LRCC on Llama and Qwen models for language modeling and zero-shot downstream tasks. Within the same average active-parameter budget, LRCC improves the predictive performance over static low-rank compression, including a 7.6 percentage-point gain in average downstream accuracy on Llama-2-7B over static methods. At matched batch-size-1 decoding latency, LRCC improves both perplexity and downstream accuracy on Llama-3.2-1B and remains competitive on Llama-2-7B, without specialized kernels. Finally, we assess the usefulness of assigning a token-wise path by analyzing the routers' path choices.

1 INTRODUCTION

Foundation models (Bommasani et al., 2021) achieve strong predictive performance across several tasks (Devlin et al., 2019; Brown et al., 2020; Radford et al., 2021) but require substantial computational and memory resources, motivating efficiency techniques. Low-rank compression methods (Chen et al., 2021) approximate the original weight matrices with low-rank products, typically obtained using SVD or its activation-aware variants. Given activations collected on a calibration dataset, these methods seek a rank-constrained approximation that minimizes the reconstruction error of each layer's outputs. Layer-specific ranks are then selected to satisfy a global budget via fixed heuristics or learned allocations (Qi et al., 2026; Qinsi et al., 2025). Conditional computation (Scardapane et al., 2024; Pfeiffer et al., 2023) instead adapts the computational graph to input activations. Applying it to pretrained models, however, can require additional training and compute modules alongside lightweight routers.

Previous work has demonstrated that SVD-like decompositions of linear layers can be used to produce many-in-one models, where the smaller ones correspond to a subset of the weights of larger ones (Yuan et al., 2025; Zaccone et al., 2026). The nested structure of these approaches provides an inexpensive basis for conditional computation. Routing over this hierarchy at every linear layer would nevertheless require many routers, each choosing among thousands of possible ranks. For example, a linear layer in Llama-2-7B can have up to 4096 rank choices, and each Transformer block contains seven such layers. The challenge is to expose useful flexibility without making routing itself expensive.

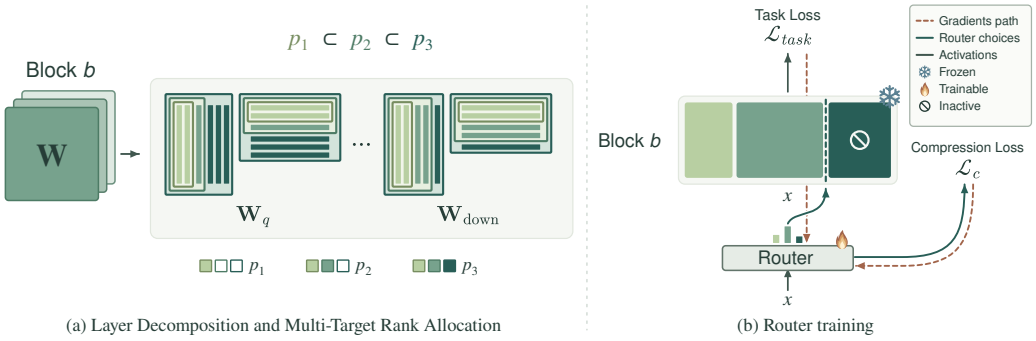
This work asks two questions: *can token-dependent rank allocation improve predictive performance within the same active-parameter budget, and do these gains translate into better quality at matched decoding latency?* We distinguish these questions because activating fewer parameters does not by itself guarantee lower latency. We address these questions by proposing **L**ow-**R**ank **C**onditional **C**omputation (LRCC). Through low-rank decomposition and rank allocation strategies, we construct a small set of models with shared weights and different computational costs. The smaller models correspond to a subset of the ranks used in the larger ones, leading to a nested family of models.

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.

A single router per Transformer block is then trained to select which subset of ranks to activate within the block, while the pretrained model and shared low-rank factors remain frozen. By using a small number of submodels and limiting the routers to the block level, we are able to show that token-dependent computation can be recovered from an existing low-rank hierarchy by training only lightweight routers, while still producing meaningful quality gains under controlled compute budgets.

Our main contributions are summarized below:

1. We introduce LRCC, which combines shared low-rank factors with block-level routing to allocate ranks dynamically to each token.
2. We show that LRCC improves predictive performance over static low-rank compression within the same active-parameter budget, with gains of up to 7.6 percentage points in average downstream accuracy. It also achieves competitive quality at matched batch-size-1 decoding latency without specialized CUDA kernels.
3. We introduce a target-cost annealing scheme that produces a family of router checkpoints in one progressive training run, and analyze how the number of paths, routing choices, and deployment costs affect the resulting models.



(a) Layer Decomposition and Multi-Target Rank Allocation

(b) Router training

Figure 1: Overview of LRCC. First, (a) each linear weight matrix is decomposed into ordered rank-one components, and a rank-allocation algorithm constructs nested paths at different target costs. Then, (b) the low-rank factors are frozen and one router per block is trained to select a path for each token, balancing task and compression losses.

2 METHOD

Conventional low-rank compression methods (Yuan et al., 2025; Wang et al., 2025; Parkina & Rakhuba, 2025; Qi et al., 2026; Qinsi et al., 2025) replace each linear weight matrix with a product of smaller matrices and retain only a fixed number of rank components in each layer. The number retained in each layer is chosen so that the compressed model meets a predefined inference-cost budget, typically expressed as a fraction of the original model's cost.

In contrast, LRCC dynamically determines the number of ranks activated in each linear layer of a Transformer block for each token. More precisely, we define multiple *paths* through each Transformer block, where each path specifies the number of ranks activated in every linear layer of the block. As in Matryoshka learning (Kusupati et al., 2022), the paths are ordered and nested: lower-cost paths retain subsets of the components used by higher-cost paths. Consequently, all paths share the same parameter base and can be selected using a single path index. We train only the block-level routers that choose among the possible paths for each token, while the remaining parameters are frozen. Figure 1 illustrates the framework.

2.1 LAYER DECOMPOSITION

For each linear weight matrix $\mathbf{W}_{b\ell} \in \mathbb{R}^{d_{\text{in}} \times d_{\text{out}}}$ at layer ℓ in Transformer block b, we compute an activation-aware decomposition using a matrix of input activations $\mathbf{X}_{b\ell} \in \mathbb{R}^{n \times d_{\text{in}}}$, where n is the

number of token activations collected on a calibration dataset. More formally, we solve for each layer

$$\arg\min_{\widetilde{\mathbf{W}}_{b\ell,r}} \left\| \mathbf{X}_{b\ell}\mathbf{W}_{b\ell} - \mathbf{X}_{b\ell}\widetilde{\mathbf{W}}_{b\ell,r} \right\|_F \quad \text{s.t.} \quad \operatorname{rank}(\widetilde{\mathbf{W}}_{b\ell,r}) \leq r. \tag{1}$$

Let $\mathbf{V}_{b\ell} \in \mathbb{R}^{d_{\text{out}} \times d_{\text{out}}}$ denote the matrix of right singular vectors of $\mathbf{X}_{b\ell}\mathbf{W}_{b\ell}$ with columns ordered by decreasing singular value, and let $\mathbf{V}_{b\ell,:r}$ denote its first r columns. Then, Equation (1) has an analytical solution given by

$$\widetilde{\mathbf{W}}_{b\ell,r} = \mathbf{W}_{b\ell}\mathbf{V}_{b\ell,:r}\mathbf{V}_{b\ell,:r}^{\top}. \tag{2}$$

Appendix B proves optimality and nestedness and Appendix E.1 describes how to compute the factors efficiently.

Equation (2) has the important property that, similar to standard SVD, optimal solutions for different target ranks r are nested. Hence, we can obtain the best rank-r activation-aware approximation by indexing the first r columns of the precomputed matrices $\mathbf{A}_{b\ell} = \mathbf{W}_{b\ell}\mathbf{V}_{b\ell}$ and $\mathbf{B}_{b\ell,} = \mathbf{V}_{b\ell}$. More concretely,

$$\widetilde{\mathbf{W}}_{b\ell,r} = (\mathbf{W}_{b\ell}\mathbf{V}_{b\ell,:r})\mathbf{V}_{b\ell,:r}^{\top} = \mathbf{A}_{b\ell,:r}\mathbf{B}_{b\ell,:r}^{\top}. \tag{3}$$

Naively replacing a weight matrix with the product of its low-rank factors does not necessarily reduce computation. A dense linear transformation requires approximately $2d_{\text{in}}d_{\text{out}}$ FLOPs per token, whereas its rank-r factorization requires approximately $2r(d_{\text{in}} + d_{\text{out}})$ FLOPs, yielding a computational benefit only when $r < \frac{d_{\text{in}}d_{\text{out}}}{d_{\text{in}}+d_{\text{out}}}$.

2.2 Path Construction

In Section 2.1, we explained how to produce the decompositions of each linear transform in the model. Crucially, the decompositions form a nested structure, where retaining more components cannot increase reconstruction error in Equation (1), but induces a larger number of FLOPs. Routing over every possible rank at every layer would introduce many routing decisions and a large choice set for each router. Our path-count ablation in Section 4.1 shows that a small set of choices is sufficient to capture most of the quality gains.

Instead, we simplify the routing problem by placing routers only at the beginning of Transformer blocks and restricting them to a small set of choices. We first construct T submodels at different target costs using the rank-allocation procedure of Swift-SVD (Qi et al., 2026). Given a target cost, Swift-SVD assigns a rank to each linear layer according to its estimated importance to model performance.

More formally, submodel m with $m = 1, ..., T$ is defined by the rank allocation

$$\mathcal{R}_m = \{r_{b\ell m}\}_{b,\ell},$$

where $r_{b\ell m}$ denotes the rank retained by linear layer ℓ in Transformer block b under submodel m. Equivalently, each weight matrix of the submodel is approximated as

$$\widetilde{\mathbf{W}}_{b\ell}^{(m)} = \mathbf{A}_{b\ell,:r_{b\ell m}}\mathbf{B}_{b\ell,:r_{b\ell m}}^{\top}. \tag{4}$$

Thus, each submodel specifies a different computational budget by assigning a particular rank to every linear layer. These whole-model allocations define the choices available to each block-level router. In block b, path p_{bm} is the restriction of submodel m's allocation to that block,

$$p_{bm} = \{r_{b\ell m}\}_{\ell}.$$

Selecting p_{bm} therefore determines simultaneously how many ranks are activated in each linear layer of block b.

We can count the number of parameters in a path based on its set of ranks. A factorized matrix of rank r activates $r(d_{\text{in}} + d_{\text{out}})$ parameters, since it is represented by two factors of sizes $d_{\text{in}} \times r$ and $d_{\text{out}} \times r$. Therefore, the parameter count of path p_{bm} is

$$P_{bm} = \sum_{\ell} r_{b\ell m} \left(d_{\text{in},b\ell} + d_{\text{out},b\ell} \right). \tag{5}$$

LRCC trains routers to assign each token to a corresponding path. Specifically, for each token entering a Transformer block b, the router $g_b : \mathbb{R}^{d_h} \to [0,1]^T$ is a learnable linear transform that maps the input activations $h \in \mathbb{R}^{d_h}$ to a Gumbel-Softmax distribution over the T available paths. During both training and inference, a hard one-hot selection determines the executed path and hence the number of activated parameters.

2.3 Router Training

The routers are optimized to minimize the task loss and a *compression loss*, which we define in this section. To preserve differentiability and retain the hard one-hot selection of the routers, we use the straight-through estimator (STE) (Jang et al., 2017). During training, all pretrained model parameters and low-rank factors remain frozen, and only the routers are updated. LRCC uses the ratio of activated parameters relative to the original model as the cost of processing a given token. Let c_b denote the average normalized cost of block b, and let C_b denote the normalized cost of its original non-factorized implementation. Because a low-rank factorization can activate more parameters than the original block, we softly cap its cost during training using a smooth approximate of $\min(c_b, C_b)$ given by

$$\bar{c}_b = c_b - \frac{C_b}{\kappa} \operatorname{softplus}\left(\frac{\kappa(c_b - C_b)}{C_b}\right), \tag{6}$$

where κ is a hyperparameter that determines the smoothness of the approximation. This reflects that blocks whose factorized implementation is more expensive than the original one should be kept non-factorized when deployed, while avoiding the zero-gradient region introduced by a hard cap.

The global cost and corresponding compression loss are then

$$c_{\text{global}} = \sum_{b=1}^{B} \bar{c}_b, \qquad \mathcal{L}_c = \max\left(0, c_{\text{global}} - t\right), \tag{7}$$

where t is the user-defined target cost. The routers are then trained to jointly minimize the task loss $\mathcal{L}_{task}$ and the computational cost:

$$\mathcal{L} = \mathcal{L}_{\text{task}} + \lambda_s \mathcal{L}_{\text{c}}, \tag{8}$$

where λ_s is an adaptive weight updated at training step s according to the current constraint violation, as described in Section 2.4. Further details on the computation of c_b and the soft cap are given in Appendix C.

2.4 Target-cost Annealing

Because the paths are nested, LRCC naturally supports producing routers for multiple target costs in a single progressive training run, moving from a high-cost model to one with fewer activated parameters. We start with a high target cost, 0.95 in our experiments, and train the router until convergence (or until a maximum number of epochs is reached). Then, we reduce the target cost to a slightly lower value, 0.925 in our case, and continue training the router weights to produce a new checkpoint. We stop when the final target cost is reached, or when convergence fails for some target, i.e., the training process failed to obtain a router inducing a cost within an acceptable distance of the current target.

As compression becomes more aggressive, the task's loss increases and can dominate the total cost, making it difficult to reach a checkpoint sufficiently close to the target. To avoid this, we introduce an adaptive compression-loss weight λ_s, which is updated after each optimization step based on the constraint violation:

$$\lambda_{s+1} = \max(0, \lambda_s + \alpha(c_{\text{global}} - t)), \tag{9}$$

where α is a hyperparameter. Intuitively, when $c_{\text{global}} > t$, the compression-loss weight increases, and when $c_{\text{global}} < t$, it decreases, allowing the router to activate more parameters.

3 Results

Models. We evaluate the performance of LRCC on Llama-3.2-1B (Meta AI, 2024), Llama-2-7B (Touvron et al., 2023) and Qwen3-4B (Yang et al., 2025) models, as they allow us to compare with the low-rank baseline methods (Qi et al., 2026; Parkina & Rakhuba, 2025; Wang et al., 2025).

Ratio	Method	PPL ↓ Wiki2	Commonsense reasoning accuracy (%) ↑ ARC-C	ARC-E	BoolQ	Hella	OBQA	PIQA	Wino	**Avg.**
100%	Base	5.49	44.80	73.95	79.42	76.08	44.00	78.62	69.30	66.60
90%	COALA	7.27	35.5 ±1.4	57.5 ±1.0	49.3 ±0.9	62.8 ±0.5	38.4 ±2.2	70.6 ±1.1	65.2 ±1.3	54.2 ±0.5
	SVD-LLM	7.27	36.2 ±1.4	57.4 ±1.0	50.1 ±0.9	62.8 ±0.5	39.0 ±2.2	70.8 ±1.1	65.3 ±1.3	54.5 ±0.5
	SVD-LLM+Swift	7.33	35.5 ±1.4	58.3 ±1.0	55.0 ±0.9	62.8 ±0.5	38.4 ±2.2	71.1 ±1.1	62.9 ±1.4	54.8 ±0.5
	COALA+Swift	7.27	35.1 ±1.4	58.4 ±1.0	57.4 ±0.9	62.8 ±0.5	38.4 ±2.2	70.9 ±1.1	63.1 ±1.4	55.1 ±0.5
	Swift-SVD	7.33	35.6 ±1.4	58.7 ±1.0	55.4 ±0.9	62.8 ±0.5	39.6 ±2.2	70.7 ±1.1	62.9 ±1.4	55.1 ±0.5
	LRCC-5	6.17	**41.8** ±1.4	68.5 ±1.0	74.4 ±0.8	71.0 ±0.5	**41.4** ±2.2	**74.8** ±1.0	66.3 ±1.3	62.6 ±0.5
	LRCC-9	**6.03**	40.4 ±1.4	**68.6** ±1.0	**74.9** ±0.8	**71.7** ±0.4	40.8 ±2.2	**74.8** ±1.0	**67.8** ±1.3	**62.7** ±0.5
70%	COALA	10.75	28.0 ±1.3	43.9 ±1.0	41.2 ±0.9	46.1 ±0.5	**35.0** ±2.1	62.1 ±1.1	59.3 ±1.4	45.1 ±0.5
	SVD-LLM	10.75	27.2 ±1.3	44.2 ±1.0	41.6 ±0.9	46.1 ±0.5	34.6 ±2.1	61.8 ±1.1	59.2 ±1.4	45.0 ±0.5
	SVD-LLM+Swift	10.58	27.7 ±1.3	44.1 ±1.0	41.9 ±0.9	46.6 ±0.5	34.8 ±2.1	61.9 ±1.1	60.5 ±1.4	45.4 ±0.5
	COALA+Swift	10.58	27.5 ±1.3	44.1 ±1.0	42.0 ±0.9	46.4 ±0.5	34.4 ±2.1	61.6 ±1.1	60.3 ±1.4	45.2 ±0.5
	Swift-SVD	10.58	28.1 ±1.3	44.2 ±1.0	43.1 ±0.9	46.5 ±0.5	**35.0** ±2.1	61.1 ±1.1	59.4 ±1.4	45.4 ±0.5
	LRCC-5	9.49	31.4 ±1.4	46.3 ±1.0	**46.6** ±0.9	56.0 ±0.5	34.6 ±2.1	**67.8** ±1.1	**61.6** ±1.4	**49.2** ±0.5
	LRCC-9	**9.05**	**31.9** ±1.4	**46.4** ±1.0	41.7 ±0.9	**56.5** ±0.5	34.4 ±2.1	65.1 ±1.1	61.5 ±1.4	48.2 ±0.5

Table 1: Predictive performance of Llama-2-7B within the indicated active-parameter budgets. LRCC achieves the highest average accuracy at all compression ratios. Results for Llama-2-7B compressed using 256 samples of sequence length 2048. Avg. reports mean accuracy over the seven reasoning tasks; Wiki2 reports perplexity.

LRCC setup. We decompose all linear layers in each Transformer block, including both attention and MLP projections, as described in Section 2.1, and compute the ranks assigned to each block path using a rank-allocation algorithm, as described in Section 2.2. In all cases, we follow previous work and use 256 2,048-token samples from the training set of WikiText-2 (Merity et al., 2017) to decompose the weights and train the router. We use the standard causal language-modeling loss as the task loss $\mathcal{L}_{\text{task}}$. Training starts at a target cost of 0.975, and we progressively obtain checkpoints down to 0.3 in steps of 0.025 using the annealing scheme described in Section 2.4. Finally, during evaluation, we use 64 WikiText-2 validation samples to estimate the average activated parameter count of each block. Blocks for which this exceeds the parameter count of the corresponding block with non-factorized linear layers are deployed in non-factorized form.

Datasets. The resulting models are evaluated on the WikiText-2 test set for language modeling and on several zero-shot downstream performance using ARC-Easy and ARC-Challenge (Clark et al., 2018), BoolQ (Clark et al., 2019), HellaSwag (Zellers et al., 2019), OpenBookQA (Mihaylov et al., 2018), PIQA (Bisk et al., 2020), and WinoGrande (Sakaguchi et al., 2020).

Static baselines. We compare LRCC with several state-of-the-art low-rank compression methods, namely SVD-LLM (Wang et al., 2025), COALA (Parkina & Rakhuba, 2025) and Swift-SVD (Qi et al., 2026). Since COALA and SVD-LLM do not propose a rank allocation method, we also evaluate variants using ranks obtained with the SWIFT-SVD rank allocation strategy. We omit SVD-LLM's optional fine-tuning stage to isolate the effect of the compression method under a matched adaptation budget, since the other baselines do not use post-compression fine-tuning of the weights and an analogous refinement stage could likewise be applied to LRCC and the baselines. See Appendix D for details and additional results.

3.1 PREDICTIVE PERFORMANCES

Table 1 shows the predictive performance results across selected ratios for Llama-2-7B (see Appendix D.1 for results on Llama-3.2-1B and Qwen3-4B). Since LRCC is dynamic, its actual average ratio of activated parameters slightly differs from the specified target costs. In cases where the actual cost was higher on a task than the target cost, we show the performance obtained using a router with lower target costs, to ensure a fairer comparison. The actual average costs for each model in each task shown are specified in Table 7. In Section 4.3, we show that the actual cost of the LRCC models follow closely their target costs across several datasets not seen during training.

At the 90% budget, LRCC-9 reaches 62.7% average accuracy, compared with 55.1% for the best static baseline, a gain of 7.6 percentage points. At the 70% budget, LRCC-5 improves average accuracy from 45.4% to 49.2%. Both variants also improve WikiText-2 perplexity at these budgets. The gains extend across most individual tasks, showcasing the beneficial effect of introducing conditional computation to the low-rank decompositions. Several static baselines obtain similar results among themselves, particularly on Llama-2-7B. This is consistent with their shared activation-aware reconstruction objective: in exact arithmetic and under the same calibration and rank allocation, their decompositions target equivalent optimal output approximations, while their numerical procedures differ. Among the evaluated combinations, COALA+Swift generally gives the strongest static results on Llama, so we use it for the trade-off curves.

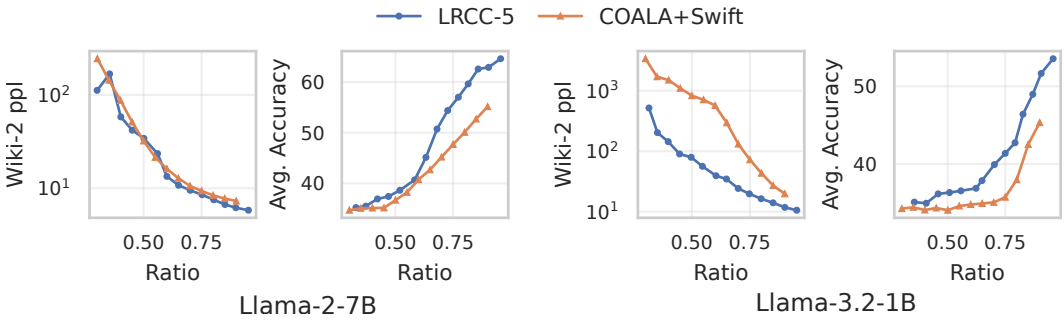


Figure 2: Predictive performance versus actual active-parameter ratio for LRCC-5 and COALA+Swift. Average accuracy is the mean over the seven downstream tasks.

Figure 2 compares LRCC-5 with COALA+Swift over a wider range of measured active-parameter ratios. We evaluate checkpoints with target costs from 0.3 to 0.95 in steps of 0.05 and plot their actual average costs on each dataset. The path-count ablation in Section 4.1 motivates using five paths and Appendix D.4 gives additional results. LRCC improves the quality–cost trade-off in almost all plotted settings, with particularly large gains on the downstream tasks. We hypothesize that aggressive static truncation removes subspaces important for particular tasks, leading to sharp performance drops (Ahmad et al., 2025). LRCC can retain such components for selected tokens while meeting an average budget, providing a possible explanation for the gains. We also observe a greater language-modeling benefit on Llama-3.2-1B. One possible explanation is that larger models are more robust to compression (Yuan et al., 2025).

To test whether the quality gains extend beyond Llama, we compare LRCC with SWIFT-SVD on Qwen3-4B. Appendix D.1 reports the results and the measured-cost curves in Figure 7. LRCC improves perplexity and average downstream accuracy across the reported budgets.

3.2 COMPUTATIONAL PERFORMANCE

We also measured the latency for models with target costs from 0.3 to 0.9 on samples from the test sets. Both LRCC and the baseline were compiled with TorchDynamo and TorchInductor using identical settings. Following Lin et al. (2024), we measured batch-size-1 autoregressive decoding, which is a setting relevant to local, resource-constrained deployment. Each measurement was repeated 5 times on a DGX Spark (NVIDIA GB10 Grace Blackwell) with 128 GB of unified memory. We teacher-forced the ground-truth task tokens through the models to ensure identical inputs for both deployments and measured latency as the time required to decode each complete sequence divided by its number of decoded tokens. For multiple-choice tasks, we timed each candidate continuation and treated all continuations associated with the same document as one sample cluster. Figure 3 shows the resulting latency–quality trade-offs. Appendix D.2 gives the latency tables and measurement protocol; Appendix D.4 reports compilation and prefill times. LRCC-5 achieves better predictive performance at the same latency on all tasks for Llama-3.2-1B and on downstream tasks for Llama-2-7B at higher latencies. The results are mixed for Llama-2-7B on WikiText-2.

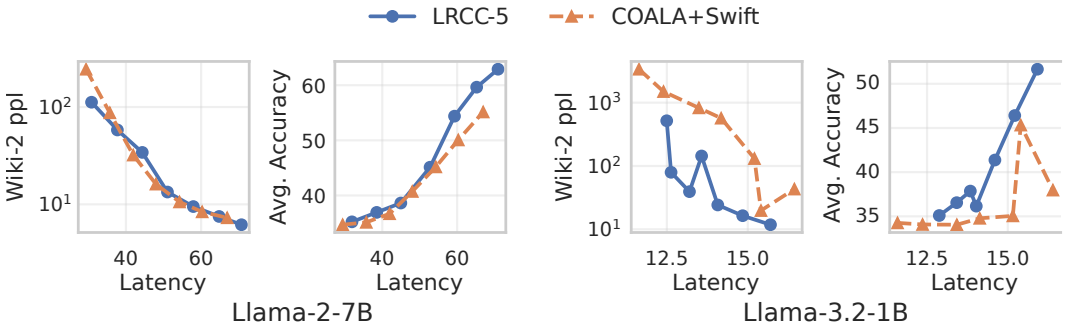


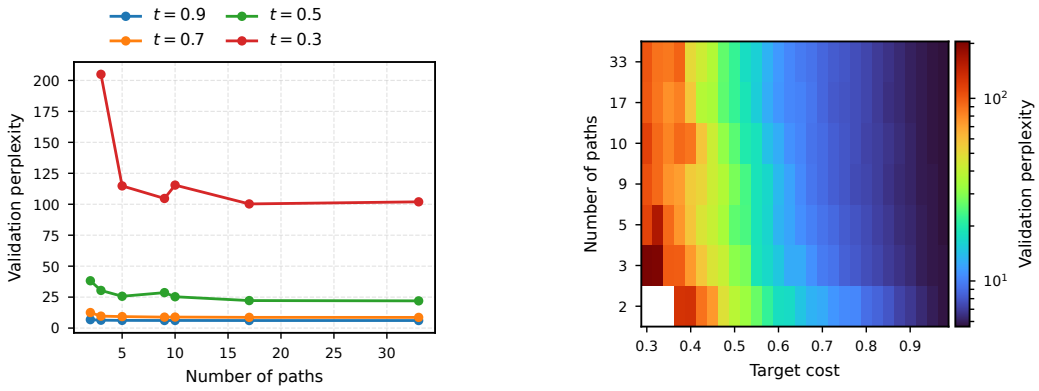
Figure 3: Pareto curves of LRCC-5 and COALA+Swift across various decoding latencies (ms/token) applied to Llama-2-7B and Llama-3.2-1B. The average accuracy is obtained by averaging the results on the commonsense reasoning downstream tasks.

4 Analysis

LRCC assigns variable amounts of compute to different input samples by choosing a different path among available options. We now ask how many paths are needed, whether the routers use them differently across tokens, and whether the learned budgets transfer beyond the router-training data.

4.1 Effect of the Number of Paths

LRCC requires determining the number of available paths within each block. To better understand the influence of this hyperparameter, we trained models with $T \in \{2, 3, 5, 9, 10, 17, 33\}$ paths per block and measured perplexity on the WikiText-2 validation set. Figure 4 shows the validation perplexity obtained across several target costs and number of paths. Perplexity increases sharply when only two paths are available and remains notably higher with three paths. The effects are particularly pronounced at lower target costs, suggesting that the limited granularity in these settings makes it harder to find a feasible solution. Increasing the number of paths beyond five provides comparatively little improvement in most settings. Additional results for Llama-3.2-1B, and Torch compilation, prefill and decoding times across various path cardinalities are shown in Appendix D.4.



(a) **(b)**

Figure 4: Ablation on the number of paths for Llama-2-7B. (a) shows WikiText-2 validation perplexity as a function of the number of paths for selected target costs, while (b) shows the results for all target costs from 0.3 to 0.975 in steps of 0.025. Hyperparameter configurations for which no run converged are shown in white.

4.2 Token-dependent Routing

To test whether the routers collapse to a single path for all tokens, we analyzed their choices for 64 samples from WikiText-2, each comprising 2,048 tokens. Specifically, we measured the average

computational cost across each block and the effective fraction of paths at different target costs. For each router, we define the effective fraction of paths as

$$F_{\text{eff}} = \frac{\exp(H(f))}{T}, \tag{10}$$

where f denotes the empirical distribution of path selections across tokens, $H(f)$ is its entropy computed with natural logarithms, and T is the number of available paths. This quantity equals 1 when all paths are selected with equal frequency and $1/T$ when all tokens are assigned to a single path. Figure 5(a) shows the results for Llama-2-7B. The results show a backbone of blocks retained

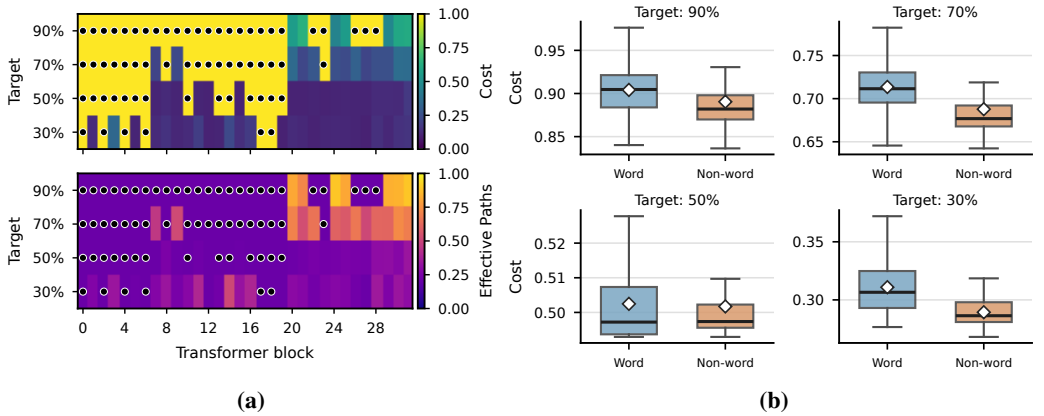


Figure 5: Router choices for Llama-2-7B on WikiText-2. (a) Average per-block cost (top) and effective fraction of paths (bottom) for each target. The dots indicate blocks that are deployed without factorization. (b) Distributions of the costs assigned to word and non-word tokens for each target.

in dense form, alongside blocks that route tokens through different low-rank paths. As the target cost decreases, the number of blocks that exhibit conditional routing increases. Multiple paths remain in use, with some Transformer blocks using almost all available paths. Together, these observations show that the routed blocks do not collectively collapse to a static rank allocation, even without an explicit path-balancing loss.

We also analyze whether certain tokens were more often assigned to low-cost paths. As shown in Figure 5(b), non-word tokens (i.e., tokens containing no alphanumeric characters) are more frequently assigned to lower-cost paths. This provides evidence that compute allocation varies with token type. Appendix D.5 reports similar patterns for Llama-3.2-1B.

4.3 Target vs. Actual Deployment Cost

LRCC routers are fitted to use a certain amount of compute on average, but this is not strictly enforced during inference. Hence, we measured the actual ratio of activated parameters of LRCC across different datasets used for testing. Figure 6 shows the results for Llama-2-7B. Although the routers were trained only on WikiText-2, actual deployment costs remain close to their targets across the evaluated datasets. Appendix D.3 presents the results for Llama-3.2-1B and Qwen3-4B.

5 Related Work

Low-rank Compression Several low-rank compression methods have been proposed in recent years (Shihab et al., 2026; Qi et al., 2026; Parkina & Rakhuba, 2025; Wang et al., 2025; Yuan et al., 2025; Qinsi et al., 2025). Zaccone et al. (2026) and Rauba & van der Schaar (2026) construct nested low-rank submodels to support multiple compute budgets within a single model. Concurrent work by Zhu et al. (2026) selects prompt-dependent singular components under fixed per-layer rank budgets and reuses the selected subspaces across decoding steps. These approaches offer either selectable global budgets or prompt-dependent subspaces. LRCC instead varies the active rank at each block for each token, under an average cost target.

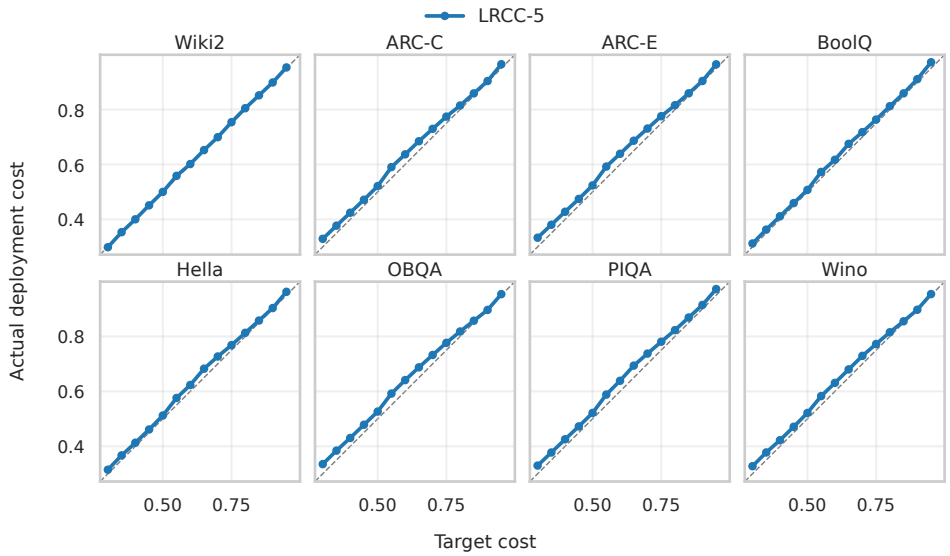


Figure 6: Target versus actual deployment cost for Llama-2-7B with LRCC-5. LRCC costs remain close to the target across different datasets.

Conditional Computation Mixture-of-Experts (Shazeer et al., 2017) has become standard in LLM pretraining, and several works have proposed architectures with dynamic computational graphs (e.g., (Raposo et al., 2024; Ainslie et al., 2023; Devvrit et al., 2024)). However, these approaches generally require training the model from scratch. Like LRCC, some approaches adapt pretrained checkpoints. Moefication and upcycling are arguably among the best known (Zhang et al., 2022; Komatsuzaki et al., 2023; Szatkowski et al., 2024), but other forms have been studied (e.g. (Wójcik et al., 2025; Cai et al., 2024)). These techniques are generally either difficult to apply to existing models (e.g., Moefication is tailored to ReLU-induced activation sparsity) or require training additional compute modules, whereas LRCC trains only lightweight routers over fixed factors.

6 Conclusion

In this work, we introduced LRCC, a technique for adding conditional computation to pretrained Transformers by training block-level routers over shared low-rank factors. LRCC improves predictive performance within fixed active-parameter budgets on Llama models, with gains of up to 7.6 percentage points in average downstream accuracy; the Qwen experiments extend the quality gains to another model family. At matched batch-size-1 decoding latency, LRCC improves perplexity and average downstream accuracy on Llama-3.2-1B and remains competitive on Llama-2-7B, without custom CUDA kernels. Our analysis shows that a small set of paths captures most of the quality gains and that the routers retain token-dependent path choices without an explicit balancing loss. Together, these results establish shared low-rank hierarchies as a practical basis for introducing conditional computation into pretrained models.

AI use statement

We used generative AI tools, including coding agents, to discuss, critique, and refine our methodological ideas, improve the exposition of proof sketches, and assist with implementing code, given our specifications of the algorithms and requirements. We also used generative AI tools for literature search and to revise text passages for clarity and presentation. AI tools were not used to independently formulate the proposed method, design or run experiments, interpret experimental results, or generate experimental claims. AI-assisted code was manually inspected, mathematical arguments were independently verified, and citations were checked against the original sources. We take responsibility for the final content of the paper and all AI-assisted work.

Ethics statement

This paper presents work whose goal is to advance the field of Machine Learning. There are many potential societal consequences of our work, none of which we feel must be specifically highlighted here.

Reproducibility statement

Section 2.1 describes the activation-aware low-rank decomposition, Section 2.2 the construction of block-level paths, Section 2.3 the router training objective, and Section 2.4 the target-cost annealing procedure. Appendix B proves the low-rank approximation result. Appendix C describes the cost computation and compression objective for training, and Appendix E reports implementation details, training settings, and baseline hyperparameters. Appendix D.2 describes the latency-measurement protocol.

References

Areeb Ahmad, Abhinav Joshi, and Ashutosh Modi. Beyond components: Singular vector-based interpretability of transformer circuits. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025. URL https://openreview.net/forum?id=7UbXEQNny7.

Joshua Ainslie, Tao Lei, Michiel de Jong, Santiago Ontanon, Siddhartha Brahma, Yury Zemlyanskiy, David Uthus, Mandy Guo, James Lee-Thorp, Yi Tay, Yun-Hsuan Sung, and Sumit Sanghai. CoLT5: Faster long-range transformers with conditional computation. In *The 2023 Conference on Empirical Methods in Natural Language Processing*, 2023. URL https://openreview.net/forum?id=h96N32OkAx.

Yonatan Bisk, Rowan Zellers, Ronan Le Bras, Jianfeng Gao, and Yejin Choi. PIQA: Reasoning about physical commonsense in natural language. *Proceedings of the AAAI Conference on Artificial Intelligence*, 34(05):7432–7439, 2020. doi: 10.1609/aaai.v34i05.6239.

Rishi Bommasani, Drew A Hudson, Ehsan Adeli, Russ Altman, Simran Arora, Sydney von Arx, Michael S Bernstein, Jeannette Bohg, Antoine Bosselut, Emma Brunskill, et al. On the opportunities and risks of foundation models. *arXiv preprint arXiv:2108.07258*, 2021.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel Ziegler, Jeffrey Wu, Clemens Winter, Chris Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin (eds.), *Advances in Neural Information Processing Systems*, volume 33, pp. 1877–1901. Curran Associates, Inc., 2020. URL https://proceedings.neurips.cc/paper_files/paper/2020/file/1457c0d6bfcb4967418bfb8ac142f64a-Paper.pdf.

Ruisi Cai, Saurav Muralidharan, Greg Heinrich, Hongxu Yin, Zhangyang Wang, Jan Kautz, and Pavlo Molchanov. Flextron: Many-in-one flexible large language model. In *Forty-first International Conference on Machine Learning*, 2024. URL https://openreview.net/forum?id=9vKRhnflAs.

Patrick Chen, Hsiang-Fu Yu, Inderjit S Dhillon, and Cho-Jui Hsieh. DRONE: Data-aware low-rank compression for large NLP models. In A. Beygelzimer, Y. Dauphin, P. Liang, and J. Wortman Vaughan (eds.), *Advances in Neural Information Processing Systems*, 2021. URL https://openreview.net/forum?id=sthiz9zeXGG.

Christopher Clark, Kenton Lee, Ming-Wei Chang, Tom Kwiatkowski, Michael Collins, and Kristina Toutanova. BoolQ: Exploring the surprising difficulty of natural yes/no questions. In Jill Burstein, Christy Doran, and Thamar Solorio (eds.), *Proceedings of the 2019 Conference of*

the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers), pp. 2924–2936, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics. doi: 10.18653/v1/N19-1300. URL https://aclanthology.org/N19-1300/.

Peter Clark, Isaac Cowhey, Oren Etzioni, Tushar Khot, Ashish Sabharwal, Carissa Schoenick, and Oyvind Tafjord. Think you have solved question answering? try ARC, the AI2 reasoning challenge. *arXiv preprint arXiv:1803.05457*, 2018.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. In Jill Burstein, Christy Doran, and Thamar Solorio (eds.), *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pp. 4171–4186, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics. doi: 10.18653/v1/N19-1423. URL https://aclanthology.org/N19-1423/.

Fnu Devvrit, Sneha Kudugunta, Aditya Kusupati, Tim Dettmers, Kaifeng Chen, Inderjit S Dhillon, Yulia Tsvetkov, Hannaneh Hajishirzi, Sham M. Kakade, Ali Farhadi, and Prateek Jain. MatFormer: Nested transformer for elastic inference. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024. URL https://openreview.net/forum?id=fYa6ezMxD5.

Alan Julian Izenman. Reduced-rank regression for the multivariate linear model. *Journal of Multivariate Analysis*, 5(2):248–264, 1975. doi: 10.1016/0047-259X(75)90042-1.

Eric Jang, Shixiang Gu, and Ben Poole. Categorical reparameterization with gumbel-softmax. In *International Conference on Learning Representations*, 2017.

Aran Komatsuzaki, Joan Puigcerver, James Lee-Thorp, Carlos Riquelme Ruiz, Basil Mustafa, Joshua Ainslie, Yi Tay, Mostafa Dehghani, and Neil Houlsby. Sparse upcycling: Training mixture-of-experts from dense checkpoints. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=T5nUQDrM4u.

Aditya Kusupati, Gantavya Bhatt, Aniket Rege, Matthew Wallingford, Aditya Sinha, Vivek Ramanujan, William Howard-Snyder, Kaifeng Chen, Sham M. Kakade, Prateek Jain, and Ali Farhadi. Matryoshka representation learning. In Alice H. Oh, Alekh Agarwal, Danielle Belgrave, and Kyunghyun Cho (eds.), *Advances in Neural Information Processing Systems*, 2022. URL https://openreview.net/forum?id=9njZa1fm35.

Ji Lin, Jiaming Tang, Haotian Tang, Shang Yang, Wei-Ming Chen, Wei-Chen Wang, Guangxuan Xiao, Xingyu Dang, Chuang Gan, and Song Han. Awq: Activation-aware weight quantization for on-device llm compression and acceleration. *Proceedings of machine learning and systems*, 6: 87–100, 2024.

Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. Pointer sentinel mixture models. In *International Conference on Learning Representations*, 2017. URL https://openreview.net/forum?id=Byj72udxe.

Meta AI. Llama 3.2 model card. https://github.com/meta-llama/llama-models/blob/main/models/llama3_2/MODEL_CARD.md, 2024.

Todor Mihaylov, Peter Clark, Tushar Khot, and Ashish Sabharwal. Can a suit of armor conduct electricity? a new dataset for open book 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. 2381–2391, Brussels, Belgium, October-November 2018. Association for Computational Linguistics. doi: 10.18653/v1/D18-1260. URL https://aclanthology.org/D18-1260/.

Uliana Parkina and Maxim Rakhuba. COALA: Numerically stable and efficient framework for context-aware low-rank approximation. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025. URL https://openreview.net/forum?id=z1wIUZtBmK.

Jonas Pfeiffer, Sebastian Ruder, Ivan Vulić, and Edoardo Maria Ponti. Modular deep learning. *arXiv preprint arXiv:2302.11529*, 2023.

Ruoling Qi, Yirui Liu, Xuaner Wu, Xiangyu Wang, Ming Li, Chen Chen, Jian Chen, Yin Chen, and Qizhen Weng. Swift-SVD: Theoretical optimality meets practical efficiency in low-rank LLM compression. In *Forty-third International Conference on Machine Learning*, 2026. URL https://openreview.net/forum?id=nAQ4h8FpdM.

Wang Qinsi, Jinghan Ke, Masayoshi Tomizuka, Kurt Keutzer, and Chenfeng Xu. Dobi-SVD: Differentiable SVD for LLM compression and some new perspectives. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=kws76i5XB8.

Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, et al. Learning transferable visual models from natural language supervision. In *International conference on machine learning*, pp. 8748–8763. PmLR, 2021.

David Raposo, Sam Ritter, Blake Richards, Timothy Lillicrap, Peter Conway Humphreys, and Adam Santoro. Mixture-of-depths: Dynamically allocating compute in transformer-based language models. *arXiv preprint arXiv:2404.02258*, 2024.

Paulius Rauba and Mihaela van der Schaar. Deep hierarchical learning with nested subspace networks for large language models. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=ymUOPsbxLi.

Keisuke Sakaguchi, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. Winogrande: An adversarial winograd schema challenge at scale. *Proceedings of the AAAI Conference on Artificial Intelligence*, 34(05):8732–8740, 2020. doi: 10.1609/aaai.v34i05.6399.

Simone Scardapane, Alessandro Baiocchi, Alessio Devoto, Valerio Marsocci, Pasquale Minervini, and Jary Pomponi. Conditional computation in neural networks: Principles and research trends. *Intelligenza Artificiale*, 18(1):175–190, 2024.

Noam Shazeer, Azalia Mirhoseini, Krzysztof Maziarz, Andy Davis, Quoc Le, Geoffrey Hinton, and Jeff Dean. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. In *International Conference on Learning Representations*, 2017. URL https://openreview.net/forum?id=B1ckMDqlg.

Ibne Farabi Shihab, Sanjeda Akter, and Anuj Sharma. Beyond variance: Knowledge-aware LLM compression via fisher-aligned subspace diagnostics. In Maria Liakata, Viviane P. Moreira, Jiajun Zhang, and David Jurgens (eds.), *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 47938–47954, San Diego, California, United States, July 2026. Association for Computational Linguistics. ISBN 979-8-89176-390-6. doi: 10.18653/v1/2026.acl-long.2214. URL https://aclanthology.org/2026.acl-long.2214/.

Filip Szatkowski, Bartosz Wójcik, Mikołaj Piórczyński, and Simone Scardapane. Exploiting activation sparsity with dense to dynamic-k mixture-of-experts conversion. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024. URL https://openreview.net/forum?id=38UFpdt3Tr.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, Dan Bikel, et al. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*, 2023.

Xin Wang, Yu Zheng, Zhongwei Wan, and Mi Zhang. SVD-LLM: Truncation-aware singular value decomposition for large language model compression. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=LNYIUouhdt.

Bartosz Wójcik, Alessio Devoto, Karol Pustelnik, Pasquale Minervini, and Simone Scardapane. Adaptive computation modules: Granular conditional computation for efficient inference. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 39, pp. 21510–21518, 2025.

An Yang, Anfeng Li, Baosong Yang, et al. Qwen3 technical report, 2025. URL https://arxiv.org/abs/2505.09388.

Zhihang Yuan, Yuzhang Shang, Yue Song, Dawei Yang, Qiang Wu, Yan Yan, and Guangyu Sun. ASVD: Activation-aware singular value decomposition for compressing large language models, 2025. URL https://arxiv.org/abs/2312.05821.

Riccardo Zaccone, Stefanos Laskaridis, Marco Ciccone, and Samuel Horváth. FlexRank: Nested low-rank knowledge decomposition for adaptive model deployment. In *Forty-third International Conference on Machine Learning*, 2026. URL https://openreview.net/forum?id=DK0kvnNelx.

Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. HellaSwag: Can a machine really finish your sentence? In Anna Korhonen, David Traum, and Lluís Màrquez (eds.), *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pp. 4791–4800, Florence, Italy, July 2019. Association for Computational Linguistics. doi: 10.18653/v1/P19-1472. URL https://aclanthology.org/P19-1472/.

Zhengyan Zhang, Yankai Lin, Zhiyuan Liu, Peng Li, Maosong Sun, and Jie Zhou. Moefication: Transformer feed-forward layers are mixtures of experts. In *Findings of the Association for Computational Linguistics: ACL 2022*, pp. 877–890, 2022.

Hengyi Zhu, Zhendong Mi, Grace Li Zhang, and Shaoyi Huang. Different prompts, different ranks: Prompt-aware dynamic rank selection for SVD-based LLM compression. *arXiv preprint arXiv:2605.08568*, 2026.

A LIMITATIONS

Our work focuses on demonstrating that conditioning the activated ranks on input activations can improve the quality–compute trade-off of low-rank compressed models, but several directions remain open. First, we have not evaluated LRCC on nested subspace models that refine the factorized weights through fine-tuning, such as FlexRank (Zaccone et al., 2026). Combining such refinement with conditional rank selection may yield further gains. Second, we demonstrated latency improvements only for batch-size-1 decoding, and future work should investigate optimized implementations for prefill and larger batch sizes. Finally, our paths are constructed before router training from a fixed set of nested rank allocations. Jointly learning the path structure and the routing policy, or exploring more flexible routing granularities, could further improve the allocation of computation across inputs.

B LOW-RANK APPROXIMATION RESULTS

This section establishes the optimality and nestedness of the activation-aware rank-r approximation used in Section 2.1, and its connection to reduced-rank regression (Izenman, 1975). Efficient computation of the factors is discussed in Appendix E.1.

Optimality and nestedness. For simplicity, we omit the block and layer subscripts. Let $\mathbf{W} \in \mathbb{R}^{d_{\text{in}} \times d_{\text{out}}}$ be a linear weight matrix and let $\mathbf{X} \in \mathbb{R}^{n \times d_{\text{in}}}$ contain its calibration inputs. We consider the rank-constrained reconstruction problem

$$\min_{\widetilde{\mathbf{W}}_r} \left\| \mathbf{X}\mathbf{W} - \mathbf{X}\widetilde{\mathbf{W}}_r \right\|_F \qquad \text{s.t.} \quad \operatorname{rank}(\widetilde{\mathbf{W}}_r) \le r. \tag{11}$$

Let

$$\mathbf{Y} = \mathbf{X}\mathbf{W} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^\top \tag{12}$$

be an SVD of the layer outputs, with singular values ordered in decreasing order, and let $\mathbf{V}_{:r}$ denote the first r columns of $\mathbf{V}$. We define

$$\widetilde{\mathbf{W}}_r = \mathbf{W}\mathbf{V}_{:r}\mathbf{V}_{:r}^\top. \tag{13}$$

Since $\operatorname{rank}(\mathbf{W}\mathbf{V}_{:r}\mathbf{V}_{:r}^\top) \le r$, this is a feasible solution to Equation 11. Its output on the calibration inputs is

$$\begin{aligned} \mathbf{X}\widetilde{\mathbf{W}}_r &= \mathbf{X}\mathbf{W}\mathbf{V}_{:r}\mathbf{V}_{:r}^\top && (14)\\ &= \mathbf{Y}\mathbf{V}_{:r}\mathbf{V}_{:r}^\top && (15)\\ &= \mathbf{U}_{:r}\mathbf{\Sigma}_{:r}\mathbf{V}_{:r}^\top = \mathbf{Y}_r, && (16) \end{aligned}$$

where $\mathbf{Y}_r$ is the truncated rank-r SVD of $\mathbf{Y}$. For any other feasible matrix $\widehat{\mathbf{W}}_r$,

$$\operatorname{rank}(\mathbf{X}\widehat{\mathbf{W}}_r) \le \operatorname{rank}(\widehat{\mathbf{W}}_r) \le r. \tag{17}$$

Therefore, by the Eckart–Young–Mirsky theorem,

$$\left\| \mathbf{X}\mathbf{W} - \mathbf{X}\widetilde{\mathbf{W}}_r \right\|_F = \|\mathbf{Y} - \mathbf{Y}_r\|_F \le \left\| \mathbf{X}\mathbf{W} - \mathbf{X}\widehat{\mathbf{W}}_r \right\|_F. \tag{18}$$

Hence, $\widetilde{\mathbf{W}}_r$ is a global optimum of Equation 11.

Moreover, define the full factors

$$\mathbf{A} = \mathbf{W}\mathbf{V}, \qquad \mathbf{B} = \mathbf{V}. \tag{19}$$

For every target rank r, the optimal approximation can then be obtained by indexing the first r columns of these same factors:

$$\widetilde{\mathbf{W}}_r = \mathbf{A}_{:r}\mathbf{B}_{:r}^\top = (\mathbf{W}\mathbf{V}_{:r})\mathbf{V}_{:r}^\top. \tag{20}$$

Thus, a single pair of ordered factors $\mathbf{A}$ and $\mathbf{B}$ simultaneously contains a globally optimal solution for every target rank. In particular, the solutions are nested, since increasing the rank from r to $r+1$ only adds the next rank-one component,

$$\widetilde{\mathbf{W}}_{r+1} = \widetilde{\mathbf{W}}_r + (\mathbf{W}\mathbf{v}_{r+1})\mathbf{v}_{r+1}^\top. \tag{21}$$

Connection to reduced-rank regression. Equation 11 can also be viewed as a special instance of reduced-rank regression (RRR) (Izenman, 1975). Given predictors $\mathbf{X}$ and multivariate responses $\mathbf{Y}$, RRR solves

$$\min_{\mathbf{C}} \|\mathbf{Y} - \mathbf{X}\mathbf{C}\|_F \qquad \text{s.t.} \quad \operatorname{rank}(\mathbf{C}) \le r. \tag{22}$$

Let $\mathbf{C}_{\mathrm{OLS}}$ be an ordinary least-squares solution and let $\widehat{\mathbf{Y}}_{\mathrm{OLS}} = \mathbf{X}\mathbf{C}_{\mathrm{OLS}}$. If $\mathbf{V}_{:r}$ contains the leading r right singular vectors of $\widehat{\mathbf{Y}}_{\mathrm{OLS}}$, a reduced-rank solution is

$$\mathbf{C}_r = \mathbf{C}_{\mathrm{OLS}}\mathbf{V}_{:r}\mathbf{V}_{:r}^{\top}. \tag{23}$$

In our setting the responses are exactly the outputs of the known pretrained linear transform, $\mathbf{Y} = \mathbf{X}\mathbf{W}$. Consequently, $\mathbf{W}$ itself is an ordinary least-squares solution with zero residual, so that $\widehat{\mathbf{Y}}_{\mathrm{OLS}} = \mathbf{X}\mathbf{W}$. Substituting $\mathbf{C}_{\mathrm{OLS}} = \mathbf{W}$ into the RRR solution therefore recovers

$$\mathbf{C}_r = \mathbf{W}\mathbf{V}_{:r}\mathbf{V}_{:r}^{\top} = \widetilde{\mathbf{W}}_r, \tag{24}$$

providing an alternative interpretation of Equation 13.

C Router Training Details

C.1 Router Training

For each token representation entering block b, the router produces a probability distribution over its T paths. During both training and inference, the forward pass selects a single path using the highest router probability. The executed path and its parameter count are computed from this hard one-hot selection, while the straight-through estimator (STE) propagates gradients through the soft probabilities during training.

To encourage the router to meet a target activated-parameter ratio $t \in [0, 1]$, we use a *compression loss* similar to Cai et al. (2024), defined over a batch of N sequences of length S as

$$\mathcal{L}_c = \max(0, c_{\mathsf{global}} - t), \tag{25}$$

where

$$c_{\mathsf{global}} = \sum_{b=1}^{B} \mathrm{softcap}_b(c_b) \tag{26}$$

is a differentiable surrogate for the global activated-parameter ratio. The blockwise soft cap approximates the cost of retaining a dense (i.e., non-factorized) block when its factorized implementation would be more expensive. We define these terms below.

Let $y_{ijbm} \in \{0, 1\}$ indicate whether token j in sequence i selects path m in block b. The normalized average cost of block b is

$$c_b = \frac{1}{NSP_0} \sum_{i=1}^{N} \sum_{j=1}^{S} \sum_{m=1}^{T} y_{ijbm} P_{bm}, \tag{27}$$

where

$$P_0 = \sum_{b=1}^{B} P_b^{\mathrm{dense}} \tag{28}$$

is the total parameter count of the original weight matrices across all Transformer blocks (the embedding matrix and language-model output head are not counted, as they are not compressed). Thus, c_b is the contribution of block b to the ratio of activated parameters relative to the original model. The normalized dense (non-factorized) cost of block b is

$$C_b = \frac{P_b^{\mathrm{dense}}}{P_0}. \tag{29}$$

The blockwise soft cap is defined as

$$\mathrm{softcap}_b(c_b) = c_b - \frac{C_b}{\kappa} \operatorname{softplus}\left(\frac{\kappa(c_b - C_b)}{C_b}\right), \tag{30}$$

where $\mathrm{softplus}(x) = \log(1 + e^x)$ and κ controls the slope. This function smoothly approximates $\min(c_b, C_b)$ and approaches C_b as the factorized block's average cost increases above the dense cost. Since $\sum_b C_b = 1$, all block costs use the original model's Transformer weights as their common denominator. A hard cap would limit the maximum cost to C_b, reflecting that inefficient blocks (i.e., where $c_b \geq C_b$) are retained in dense form at deployment. However, it would also create a zero-gradient region, preventing the compression loss from providing an optimization signal. The soft cap instead retains gradients above the dense cost, but is slightly lower than the hard-capped cost. We therefore report actual deployment costs separately in Sections 4.3 and D.3.

During training, we freeze all model parameters except the routers and minimize the sum of the task and compression losses

$$\mathcal{L} = \mathcal{L}_{\text{task}} + \lambda_s \mathcal{L}_{\text{c}}, \tag{31}$$

where λ_s is an adaptive compression-loss weight, updated at each training step s according to the constraint violation as described in Section 2.4. The target-cost schedule and training hyperparameters are given in Appendix E.

D Additional Experiments

D.1 Additional Predictive Performance Results

Similarly to the results presented in Section 3.1, we conduct an analysis of the downstream performance of LRCC compared to COALA, SVD-LLM, SVD-LLM, SVD-LLM+Swift, COALA+Swift, Swift-SVD (see Section 3.1 for more details). We present the complete results for Llama-2-7B, Llama-3.2-1B and Qwen3-4B respectively in Table 2, Table 3 and Table 4. As can be observed from the two tables, LRCC-5 and LRCC-9 continue to surpass static methods in nearly all evaluations. This demonstrates that models benefit from dynamically allocating different ranks depending on the tokens' needed accuracy, highlighting the strong assumptions static approaches do about models' computations.

In Figure 7 we compare LRCC-5 with SWIFT-SVD on Qwen3-4B across measured active-parameter ratios. LRCC achieves lower perplexity at every ratio, with the largest gap under aggressive compression, where SWIFT-SVD degrades sharply. On the downstream tasks, LRCC matches or exceeds SWIFT-SVD throughout, with the largest gains at intermediate ratios. These trends mirror the Llama results in Figure 2, suggesting that the benefits of LRCC are not specific to one model family.

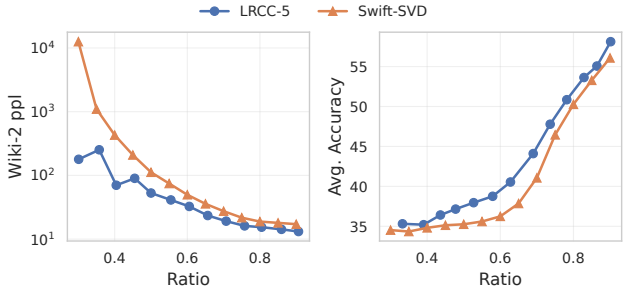


Figure 7: Pareto curves of LRCC-5 and SWIFT-SVDacross various cost ratios applied to Qwen3-4B. The average accuracy is obtained by averaging the results on the commonsense reasoning downstream tasks.

D.2 Computational Performance Results

This section provides the measurement protocol and per-task latency results for Section 3.2. We measured the latency of LRCC compared with COALA+Swift, a representative low-rank compression approach, across several target costs and downstream tasks. We teacher-forced the ground-truth task tokens through the model to ensure identical inputs for both deployments and measured latency as the time required to decode each complete sequence divided by its number of decoded tokens. For

Ratio	Method	PPL ↓ Wiki2	Commonsense reasoning accuracy (%) ↑ ARC-C	ARC-E	BoolQ	Hella	OBQA	PIQA	Wino	**Avg.**
100%	Base	5.49	44.80	73.95	79.42	76.08	44.00	78.62	69.30	66.60
90%	COALA	7.27	35.5 ±1.4	57.5 ±1.0	49.3 ±0.9	62.8 ±0.5	38.4 ±2.2	70.6 ±1.1	65.2 ±1.3	54.2 ±0.5
	SVD-LLM	7.27	36.2 ±1.4	57.4 ±1.0	50.1 ±0.9	62.8 ±0.5	39.0 ±2.2	70.8 ±1.1	65.3 ±1.3	54.5 ±0.5
	SVD-LLM+Swift	7.33	35.5 ±1.4	58.3 ±1.0	55.0 ±0.9	62.8 ±0.5	38.4 ±2.2	71.1 ±1.1	62.9 ±1.4	54.8 ±0.5
	COALA+Swift	7.27	35.1 ±1.4	58.4 ±1.0	57.4 ±0.9	62.8 ±0.5	38.4 ±2.2	70.9 ±1.1	63.1 ±1.4	55.1 ±0.5
	Swift-SVD	7.33	35.6 ±1.4	58.7 ±1.0	55.4 ±0.9	62.8 ±0.5	39.6 ±2.2	70.7 ±1.1	62.9 ±1.4	55.1 ±0.5
	LRCC-5	6.17	**41.8** ±1.4	68.5 ±1.0	74.4 ±0.8	71.0 ±0.5	**41.4** ±2.2	**74.8** ±1.0	66.3 ±1.3	62.6 ±0.5
	LRCC-9	**6.03**	40.4 ±1.4	**68.6** ±1.0	**74.9** ±0.8	**71.7** ±0.4	40.8 ±2.2	**74.8** ±1.0	**67.8** ±1.3	**62.7** ±0.5
70%	COALA	10.75	28.0 ±1.3	43.9 ±1.0	41.2 ±0.9	46.1 ±0.5	**35.0** ±2.1	62.1 ±1.1	59.3 ±1.4	45.1 ±0.5
	SVD-LLM	10.75	27.2 ±1.3	44.2 ±1.0	41.6 ±0.9	46.1 ±0.5	34.6 ±2.1	61.8 ±1.1	59.2 ±1.4	45.0 ±0.5
	SVD-LLM+Swift	10.58	27.7 ±1.3	44.1 ±1.0	41.9 ±0.9	46.6 ±0.5	34.8 ±2.1	61.9 ±1.1	60.5 ±1.4	45.4 ±0.5
	COALA+Swift	10.58	27.5 ±1.3	44.1 ±1.0	42.0 ±0.9	46.4 ±0.5	34.4 ±2.1	61.6 ±1.1	60.3 ±1.4	45.2 ±0.5
	Swift-SVD	10.58	28.1 ±1.3	44.2 ±1.0	43.1 ±0.9	46.5 ±0.5	**35.0** ±2.1	61.1 ±1.1	59.4 ±1.4	45.4 ±0.5
	LRCC-5	9.49	31.4 ±1.4	46.3 ±1.0	**46.6** ±0.9	56.0 ±0.5	34.6 ±2.1	**67.8** ±1.1	**61.6** ±1.4	**49.2** ±0.5
	LRCC-9	**9.05**	**31.9** ±1.4	**46.4** ±1.0	41.7 ±0.9	**56.5** ±0.5	34.4 ±2.1	65.1 ±1.1	61.5 ±1.4	48.2 ±0.5
50%	COALA	33.12	22.7 ±1.2	30.9 ±0.9	**37.8** ±0.8	31.0 ±0.5	26.6 ±2.0	53.2 ±1.2	53.9 ±1.4	36.6 ±0.5
	SVD-LLM	33.64	23.0 ±1.2	30.9 ±0.9	**37.8** ±0.8	31.1 ±0.5	27.0 ±2.0	53.0 ±1.2	54.5 ±1.4	36.8 ±0.5
	SVD-LLM+Swift	**32.10**	23.0 ±1.2	30.8 ±0.9	**37.8** ±0.8	31.3 ±0.5	26.6 ±2.0	53.1 ±1.2	54.7 ±1.4	36.8 ±0.5
	COALA+Swift	**32.10**	23.0 ±1.2	30.5 ±0.9	**37.8** ±0.8	31.4 ±0.5	26.6 ±2.0	52.8 ±1.2	54.6 ±1.4	36.7 ±0.5
	Swift-SVD	32.60	23.1 ±1.2	30.9 ±0.9	**37.8** ±0.8	31.4 ±0.5	27.0 ±2.0	52.4 ±1.2	**55.1** ±1.4	36.8 ±0.5
	LRCC-5	41.86	**24.8** ±1.3	32.2 ±1.0	**37.8** ±0.8	**32.7** ±0.5	27.8 ±2.0	55.3 ±1.2	52.9 ±1.4	37.6 ±0.5
	LRCC-9	41.86	22.8 ±1.2	**33.2** ±1.0	**37.8** ±0.8	32.1 ±0.5	**30.0** ±2.1	**56.1** ±1.2	53.0 ±1.4	**37.8** ±0.5

Table 2: Performance of Llama-2-7B compressed using 256 samples with sequence length 2048 on a language modeling dataset (measured using perplexity) and seven commonsense reasoning datasets (measured using accuracy).

multiple-choice tasks, we timed each candidate continuation and treated all continuations associated with the same document as one sample cluster. All measurements used a batch size of 1.

The latency estimates are based on four randomly selected 2,048-token chunks from WikiText-2, 16 documents from WinoGrande, eight from OpenBookQA, and four from each remaining task: ARC-Challenge, ARC-Easy, BoolQ, HellaSwag, and PIQA. These sample counts were selected based on pilot validation runs, with additional samples allocated to datasets exhibiting greater document-level variability. Paired document-cluster bootstrap intervals for the final workloads had relative half-widths of at most 2.14%. All candidate continuations from a sampled document were kept together during resampling to avoid treating correlated choices as independent observations.

Both methods were compiled using TorchDynamo and TorchInductor with identical settings. CUDA graphs were disabled because we found their support limited for conditional methods and to ensure a fair comparison. We performed three 64-token warm-up runs before each measurement to exclude compilation and initialization from the timed region (compilation and prefill times are explored in Appendix D.4). Experiments were conducted on an NVIDIA GB10 Grace Blackwell with 128 GB of unified memory.

We repeated each complete experiment five times to estimate run-to-run variability. Each table entry reports the mean latency in milliseconds per token across the five executions, and the subscripted $\pm$ term gives the half-width of a 95% Student-t confidence interval computed from the five execution-level measurements ($n = 5$, $df = 4$). Table 5 and Table 6 show the results.

At the same nominal target cost, LRCC is slower on Llama-2-7B, while the ordering varies with the target cost on Llama-3.2-1B. LRCC introduces overhead from evaluating the router and dispatching the selected computational path, but its executed cost also depends on the routing decisions. As discussed in Section 3, at matched latencies, LRCC produces competitive predictive performance

Ratio	Method	PPL ↓ Wiki2	Commonsense reasoning accuracy (%) ↑ ARC-C	ARC-E	BoolQ	Hella	OBQA	PIQA	Wino	**Avg.**
100%	Base	9.79	36.69	61.95	63.61	64.23	37.00	74.65	61.01	57.02
90%	COALA	22.76	25.9 ±1.3	38.4 ±1.0	48.3 ±0.9	35.0 ±0.5	27.0 ±2.0	56.4 ±1.2	52.6 ±1.4	40.5 ±0.5
	SVD-LLM	23.12	25.6 ±1.3	38.7 ±1.0	47.6 ±0.9	34.8 ±0.5	27.6 ±2.0	56.4 ±1.2	52.3 ±1.4	40.4 ±0.5
	SVD-LLM+Swift	21.05	26.7 ±1.3	41.3 ±1.0	54.3 ±0.9	38.9 ±0.5	28.2 ±2.0	59.4 ±1.1	55.5 ±1.4	43.5 ±0.5
	COALA+Swift	19.77	26.5 ±1.3	44.4 ±1.0	**54.6** ±0.9	42.1 ±0.5	31.2 ±2.1	62.2 ±1.1	56.2 ±1.4	45.3 ±0.5
	Swift-SVD	21.05	25.7 ±1.3	38.8 ±1.0	51.0 ±0.9	36.7 ±0.5	27.6 ±2.0	57.7 ±1.2	52.6 ±1.4	41.4 ±0.5
	LRCC-5	11.81	**32.6** ±1.4	**54.2** ±1.0	49.6 ±0.9	51.5 ±0.5	34.2 ±2.1	70.3 ±1.1	**56.7** ±1.4	**49.9** ±0.5
	LRCC-9	**11.62**	31.3 ±1.4	53.1 ±1.0	46.3 ±0.9	**52.5** ±0.5	**35.6** ±2.1	**70.5** ±1.1	54.7 ±1.4	49.1 ±0.5
70%	COALA	209.30	22.1 ±1.2	28.4 ±0.9	37.8 ±0.8	27.7 ±0.4	25.4 ±1.9	52.5 ±1.2	50.4 ±1.4	34.9 ±0.5
	SVD-LLM	222.79	22.4 ±1.2	28.2 ±0.9	37.8 ±0.8	28.0 ±0.4	25.0 ±1.9	52.0 ±1.2	49.7 ±1.4	34.7 ±0.5
	SVD-LLM+Swift	135.13	22.7 ±1.2	29.0 ±0.9	37.8 ±0.8	28.1 ±0.4	24.8 ±1.9	53.0 ±1.2	51.1 ±1.4	35.2 ±0.5
	COALA+Swift	130.97	22.4 ±1.2	28.8 ±0.9	37.8 ±0.8	28.1 ±0.4	24.8 ±1.9	52.6 ±1.2	50.8 ±1.4	35.1 ±0.5
	Swift-SVD	157.98	21.6 ±1.2	28.5 ±0.9	37.8 ±0.8	28.0 ±0.4	25.0 ±1.9	52.3 ±1.2	51.5 ±1.4	35.0 ±0.5
	LRCC-5	24.23	22.7 ±1.2	37.8 ±1.0	38.6 ±0.9	**35.3** ±0.5	27.4 ±2.0	58.2 ±1.2	50.3 ±1.4	38.6 ±0.5
	LRCC-9	**23.48**	**24.9** ±1.3	**38.7** ±1.0	**42.0** ±0.9	34.3 ±0.5	**28.2** ±2.0	**59.6** ±1.1	**51.9** ±1.4	**40.0** ±0.5
50%	COALA	998.50	22.6 ±1.2	25.6 ±0.9	37.8 ±0.8	27.0 ±0.4	25.6 ±2.0	49.6 ±1.2	50.4 ±1.4	34.1 ±0.5
	SVD-LLM	967.78	22.9 ±1.2	25.8 ±0.9	37.8 ±0.8	27.0 ±0.4	25.2 ±1.9	49.9 ±1.2	50.3 ±1.4	34.1 ±0.5
	SVD-LLM+Swift	854.06	**23.9** ±1.2	26.1 ±0.9	37.8 ±0.8	26.7 ±0.4	24.4 ±1.9	50.1 ±1.2	49.3 ±1.4	34.1 ±0.5
	COALA+Swift	827.78	23.4 ±1.2	26.2 ±0.9	37.8 ±0.8	26.7 ±0.4	24.6 ±1.9	50.1 ±1.2	49.5 ±1.4	34.1 ±0.5
	Swift-SVD	938.00	23.3 ±1.2	26.3 ±0.9	37.8 ±0.8	26.5 ±0.4	24.8 ±1.9	51.1 ±1.2	50.1 ±1.4	34.3 ±0.5
	LRCC-5	**79.44**	21.8 ±1.2	**33.9** ±1.0	37.8 ±0.8	**29.2** ±0.5	**26.8** ±2.0	53.8 ±1.2	**50.6** ±1.4	**36.3** ±0.5
	LRCC-9	229.87	21.2 ±1.2	33.7 ±1.0	**37.9** ±0.8	27.8 ±0.4	25.6 ±2.0	**54.5** ±1.2	49.9 ±1.4	35.8 ±0.5

Table 3: Performance of Llama-3.2-1B compressed using 256 samples with sequence length 2048 on a language modeling dataset (measured using perplexity) and seven commonsense reasoning datasets (measured using accuracy).

Ratio	Method	PPL ↓ Wiki2	Commonsense reasoning accuracy (%) ↑ ARC-C	ARC-E	BoolQ	Hella	OBQA	PIQA	Wino	**Avg.**
90%	Swift-SVD	17.18	38.1 ±1.4	56.4 ±1.0	76.2 ±0.7	58.1 ±0.5	34.8 ±2.1	68.4 ±1.1	60.7 ±1.4	56.1 ±0.5
	LRCC-5	13.17	41.1 ±1.4	63.5 ±1.0	**77.3** ±0.7	58.0 ±0.5	**35.2** ±2.1	70.3 ±1.1	**61.3** ±1.4	**58.1** ±0.5
	LRCC-9	**12.77**	**42.7** ±1.4	**65.5** ±1.0	68.8 ±0.8	**59.0** ±0.5	**35.2** ±2.1	**71.7** ±1.1	59.9 ±1.4	57.5 ±0.5
70%	Swift-SVD	27.45	25.0 ±1.3	37.9 ±1.0	48.5 ±0.9	37.1 ±0.5	29.2 ±2.0	57.9 ±1.2	51.9 ±1.4	41.1 ±0.5
	LRCC-5	19.17	27.9 ±1.3	46.5 ±1.0	**60.8** ±0.9	**44.9** ±0.5	30.8 ±2.1	62.7 ±1.1	**53.0** ±1.4	**46.7** ±0.5
	LRCC-9	**18.00**	**29.1** ±1.3	**48.1** ±1.0	45.9 ±0.9	42.2 ±0.5	**34.4** ±2.1	**65.2** ±1.1	52.7 ±1.4	45.4 ±0.5
50%	Swift-SVD	112.03	**23.9** ±1.2	28.2 ±0.9	**38.2** ±0.8	29.4 ±0.5	24.0 ±1.9	52.9 ±1.2	50.3 ±1.4	35.3 ±0.5
	LRCC-5	52.92	23.6 ±1.2	**36.4** ±1.0	37.8 ±0.8	32.4 ±0.5	**27.4** ±2.0	**55.5** ±1.2	**51.2** ±1.4	**37.8** ±0.5
	LRCC-9	**45.26**	21.9 ±1.2	32.9 ±1.0	37.8 ±0.8	**32.5** ±0.5	25.0 ±1.9	54.0 ±1.2	50.9 ±1.4	36.4 ±0.5

Table 4: Performance of Qwen3-4B compressed using 256 samples with sequence length 2048 on a language modeling dataset (measured using perplexity) and seven commonsense reasoning datasets (measured using accuracy).

and outperforms the baseline in several settings. Furthermore, our implementation does not use specialized fused CUDA kernels, leaving opportunities for further speedups. For example, fusing the router with layer normalization could reduce the number of times the residual stream is read from memory.

Target	Method	Wiki2	ARC-C	ARC-E	BoolQ	Hella	OBQA	PIQA	Wino
90%	COALA+	$66.98_{\pm 0.08}$	$67.01_{\pm 0.09}$	$67.01_{\pm 0.09}$	$66.99_{\pm 0.08}$	$67.00_{\pm 0.09}$	$67.02_{\pm 0.10}$	$67.01_{\pm 0.09}$	$67.01_{\pm 0.09}$
	LRCC-5	$70.83_{\pm 0.13}$	$70.96_{\pm 0.12}$	$70.98_{\pm 0.12}$	$71.32_{\pm 0.13}$	$71.51_{\pm 0.13}$	$70.54_{\pm 0.11}$	$71.37_{\pm 0.12}$	$70.36_{\pm 0.11}$
80%	COALA+	$60.29_{\pm 0.10}$	$60.32_{\pm 0.11}$	$60.33_{\pm 0.11}$	$60.31_{\pm 0.10}$	$60.32_{\pm 0.10}$	$60.33_{\pm 0.11}$	$60.35_{\pm 0.15}$	$60.33_{\pm 0.11}$
	LRCC-5	$64.73_{\pm 0.10}$	$64.86_{\pm 0.11}$	$65.10_{\pm 0.11}$	$65.00_{\pm 0.10}$	$65.34_{\pm 0.11}$	$65.21_{\pm 0.11}$	$65.71_{\pm 0.11}$	$65.01_{\pm 0.11}$
70%	COALA+	$54.31_{\pm 0.07}$	$54.34_{\pm 0.07}$	$54.37_{\pm 0.06}$	$54.35_{\pm 0.06}$	$54.34_{\pm 0.06}$	$54.38_{\pm 0.07}$	$54.34_{\pm 0.06}$	$54.36_{\pm 0.07}$
	LRCC-5	$57.96_{\pm 0.04}$	$59.19_{\pm 0.07}$	$59.53_{\pm 0.07}$	$58.54_{\pm 0.06}$	$59.25_{\pm 0.06}$	$59.54_{\pm 0.08}$	$60.30_{\pm 0.07}$	$59.15_{\pm 0.07}$
60%	COALA+	$48.09_{\pm 0.07}$	$48.13_{\pm 0.09}$	$48.14_{\pm 0.10}$	$48.12_{\pm 0.08}$	$48.12_{\pm 0.08}$	$48.12_{\pm 0.08}$	$48.12_{\pm 0.08}$	$48.12_{\pm 0.08}$
	LRCC-5	$51.05_{\pm 0.05}$	$53.06_{\pm 0.06}$	$53.14_{\pm 0.06}$	$51.87_{\pm 0.05}$	$52.56_{\pm 0.05}$	$53.12_{\pm 0.07}$	$53.37_{\pm 0.06}$	$52.34_{\pm 0.06}$
50%	COALA+	$41.89_{\pm 0.04}$	$41.91_{\pm 0.05}$	$41.91_{\pm 0.05}$	$41.92_{\pm 0.05}$	$41.92_{\pm 0.05}$	$41.89_{\pm 0.05}$	$41.90_{\pm 0.04}$	$41.89_{\pm 0.04}$
	LRCC-5	$44.39_{\pm 0.06}$	$45.31_{\pm 0.11}$	$45.28_{\pm 0.09}$	$44.74_{\pm 0.08}$	$45.00_{\pm 0.09}$	$44.64_{\pm 0.11}$	$45.43_{\pm 0.10}$	$44.85_{\pm 0.11}$
40%	COALA+	$35.76_{\pm 0.08}$	$35.79_{\pm 0.07}$	$35.78_{\pm 0.07}$	$35.80_{\pm 0.07}$	$35.79_{\pm 0.07}$	$35.76_{\pm 0.07}$	$35.78_{\pm 0.06}$	$35.77_{\pm 0.07}$
	LRCC-5	$37.73_{\pm 0.08}$	$38.63_{\pm 0.10}$	$38.88_{\pm 0.10}$	$38.30_{\pm 0.09}$	$38.41_{\pm 0.10}$	$38.12_{\pm 0.12}$	$39.02_{\pm 0.10}$	$38.12_{\pm 0.12}$
30%	COALA+	$29.35_{\pm 0.08}$	$29.41_{\pm 0.02}$	$29.41_{\pm 0.02}$	$29.42_{\pm 0.02}$	$29.42_{\pm 0.02}$	$29.41_{\pm 0.03}$	$29.41_{\pm 0.02}$	$29.41_{\pm 0.03}$
	LRCC-5	$30.82_{\pm 0.08}$	$32.08_{\pm 0.09}$	$32.43_{\pm 0.10}$	$31.58_{\pm 0.08}$	$31.81_{\pm 0.09}$	$31.39_{\pm 0.11}$	$32.69_{\pm 0.17}$	$31.72_{\pm 0.12}$

Table 5: Decode latency (ms/token) for LRCC-5 and COALA+Swift (shown as COALA+) at matched target costs for Llama-2-7B at batch size 1. Each value corresponds to the mean latency across five benchmark executions. We also show the 95% Student-t confidence interval half-width ($n = 5$, $df = 4$). Both methods use identical tokenized inputs in each execution.

Target	Method	Wiki2	ARC-C	ARC-E	BoolQ	Hella	OBQA	PIQA	Wino
90%	COALA+	$15.34_{\pm 0.07}$	$15.37_{\pm 0.07}$	$15.35_{\pm 0.07}$	$15.36_{\pm 0.07}$	$15.36_{\pm 0.07}$	$15.35_{\pm 0.07}$	$15.35_{\pm 0.07}$	$15.35_{\pm 0.07}$
	LRCC-5	$15.77_{\pm 0.09}$	$15.98_{\pm 0.07}$	$15.94_{\pm 0.07}$	$15.90_{\pm 0.07}$	$15.97_{\pm 0.07}$	$16.02_{\pm 0.06}$	$16.03_{\pm 0.07}$	$15.96_{\pm 0.06}$
80%	COALA+	$16.37_{\pm 0.07}$	$16.34_{\pm 0.06}$	$16.33_{\pm 0.06}$	$16.38_{\pm 0.06}$	$16.36_{\pm 0.06}$	$16.28_{\pm 0.06}$	$16.32_{\pm 0.06}$	$16.29_{\pm 0.06}$
	LRCC-5	$14.90_{\pm 0.04}$	$15.21_{\pm 0.05}$	$15.27_{\pm 0.05}$	$15.09_{\pm 0.05}$	$15.18_{\pm 0.05}$	$15.47_{\pm 0.04}$	$15.27_{\pm 0.04}$	$15.40_{\pm 0.04}$
70%	COALA+	$15.15_{\pm 0.07}$	$15.12_{\pm 0.07}$	$15.11_{\pm 0.07}$	$15.15_{\pm 0.07}$	$15.13_{\pm 0.07}$	$15.06_{\pm 0.07}$	$15.10_{\pm 0.07}$	$15.06_{\pm 0.07}$
	LRCC-5	$14.18_{\pm 0.11}$	$14.61_{\pm 0.11}$	$14.62_{\pm 0.11}$	$14.42_{\pm 0.11}$	$14.53_{\pm 0.11}$	$15.06_{\pm 0.12}$	$14.82_{\pm 0.11}$	$14.74_{\pm 0.12}$
60%	COALA+	$14.13_{\pm 0.08}$	$14.10_{\pm 0.08}$	$14.10_{\pm 0.08}$	$14.14_{\pm 0.08}$	$14.13_{\pm 0.08}$	$14.05_{\pm 0.08}$	$14.09_{\pm 0.08}$	$14.05_{\pm 0.08}$
	LRCC-5	$13.22_{\pm 0.05}$	$13.84_{\pm 0.07}$	$13.96_{\pm 0.07}$	$13.57_{\pm 0.07}$	$13.64_{\pm 0.07}$	$14.40_{\pm 0.07}$	$14.00_{\pm 0.07}$	$13.92_{\pm 0.07}$
50%	COALA+	$13.41_{\pm 0.09}$	$13.36_{\pm 0.08}$	$13.35_{\pm 0.08}$	$13.41_{\pm 0.08}$	$13.39_{\pm 0.08}$	$13.28_{\pm 0.08}$	$13.34_{\pm 0.08}$	$13.28_{\pm 0.08}$
	LRCC-5	$12.68_{\pm 0.04}$	$13.55_{\pm 0.05}$	$13.51_{\pm 0.05}$	$13.03_{\pm 0.04}$	$13.17_{\pm 0.05}$	$14.03_{\pm 0.07}$	$13.76_{\pm 0.06}$	$13.38_{\pm 0.07}$
40%	COALA+	$12.41_{\pm 0.08}$	$12.38_{\pm 0.08}$	$12.37_{\pm 0.08}$	$12.41_{\pm 0.08}$	$12.40_{\pm 0.08}$	$12.32_{\pm 0.08}$	$12.36_{\pm 0.08}$	$12.33_{\pm 0.08}$
	LRCC-5	$13.56_{\pm 0.05}$	$14.12_{\pm 0.05}$	$14.23_{\pm 0.05}$	$13.78_{\pm 0.07}$	$13.74_{\pm 0.06}$	$14.40_{\pm 0.07}$	$14.31_{\pm 0.07}$	$13.89_{\pm 0.06}$
30%	COALA+	$11.59_{\pm 0.04}$	$11.55_{\pm 0.04}$	$11.54_{\pm 0.04}$	$11.59_{\pm 0.04}$	$11.57_{\pm 0.04}$	$11.48_{\pm 0.04}$	$11.53_{\pm 0.04}$	$11.49_{\pm 0.04}$
	LRCC-5	$12.55_{\pm 0.05}$	$12.95_{\pm 0.06}$	$13.10_{\pm 0.05}$	$12.73_{\pm 0.06}$	$12.69_{\pm 0.07}$	$13.17_{\pm 0.06}$	$13.07_{\pm 0.05}$	$12.85_{\pm 0.07}$

Table 6: Decode latency (ms/token) for LRCC-5 and COALA+Swift (shown as COALA+) at matched target costs for Llama-3.2-1B at batch size 1. Each value corresponds to the mean latency across five benchmark executions. We also show the 95% Student-t confidence interval half-width ($n = 5$, $df = 4$). Both methods use identical tokenized inputs in each execution.

D.3 Additional Results on Target vs. Actual Deployment Cost

Due to the input-dependence of the LRCCmethod, the average ratio of activated parameters is not strictly the same as the specified in the target cost. Table 7 shows the actual ratio of activated parameters in the different datasets, corresponding to the same checkpoints whose results were reported in Appendix D and for Llama-2-7B in Table 1. Additionally, Figure 8 and Figure 9 show the actual deployment costs versus the target cost used during training for Llama-3.2-1B and Qwen3-4B, similarly to Section 4.3.

D.4 Ablation on the Number of Paths

As described in Section 4.1, we trained models with $P \in \{2, 3, 5, 9, 10, 17, 33\}$ paths per block to understand the influence of this hyperparameter. Each path corresponds to a rank assignment with a target cost uniformly spaced between 0.1 and 1.0. At each refinement, we retain the previous paths and add paths with target costs midway between them. For example, with $P = 2$, we use paths constructed by the rank-allocation algorithm with target costs of 0.1 and 1.0. For $P = 3$, we add

Ratio	Method	Model	Wiki2	ARC-C	ARC-E	BoolQ	Hella	OBQA	PIQA	Wino	Avg.
90%	LRCC-5	Llama-2-7B	0.90	0.86	0.86	0.86	0.86	0.90	0.87	0.90	0.87
	LRCC-9	Llama-2-7B	0.90	0.87	0.87	0.87	0.87	0.87	0.88	0.87	0.87
	LRCC-5	Llama-3.2-1B	0.90	0.87	0.88	0.87	0.87	0.90	0.88	0.89	0.88
	LRCC-9	Llama-3.2-1B	0.90	0.86	0.86	0.86	0.86	0.88	0.87	0.87	0.87
	LRCC-5	Qwen3-4B	0.91	0.90	0.90	0.91	0.90	0.90	0.91	0.89	0.90
	LRCC-9	Qwen3-4B	0.91	0.91	0.91	0.91	0.91	0.91	0.91	0.91	0.91
70%	LRCC-5	Llama-2-7B	0.70	0.68	0.69	0.67	0.68	0.69	0.69	0.68	0.68
	LRCC-9	Llama-2-7B	0.70	0.70	0.70	0.67	0.68	0.70	0.66	0.69	0.68
	LRCC-5	Llama-3.2-1B	0.70	0.65	0.65	0.68	0.68	0.68	0.65	0.64	0.66
	LRCC-9	Llama-3.2-1B	0.70	0.70	0.66	0.68	0.68	0.69	0.65	0.70	0.68
	LRCC-5	Qwen3-4B	0.71	0.69	0.69	0.72	0.72	0.71	0.70	0.70	0.70
	LRCC-9	Qwen3-4B	0.71	0.69	0.69	0.73	0.67	0.71	0.70	0.69	0.70
50%	LRCC-5	Llama-2-7B	0.45	0.47	0.47	0.46	0.46	0.48	0.47	0.47	0.47
	LRCC-9	Llama-2-7B	0.45	0.47	0.47	0.46	0.46	0.48	0.47	0.47	0.47
	LRCC-5	Llama-3.2-1B	0.50	0.47	0.47	0.48	0.47	0.50	0.47	0.46	0.47
	LRCC-9	Llama-3.2-1B	0.45	0.49	0.49	0.46	0.46	0.45	0.49	0.49	0.48
	LRCC-5	Qwen3-4B	0.50	0.52	0.53	0.51	0.51	0.50	0.48	0.49	0.51
	LRCC-9	Qwen3-4B	0.50	0.49	0.49	0.51	0.51	0.50	0.49	0.48	0.50

Table 7: Actual deployment cost (activated-parameter ratio) for the LRCC-5 and LRCC-9 configurations used during evaluation.

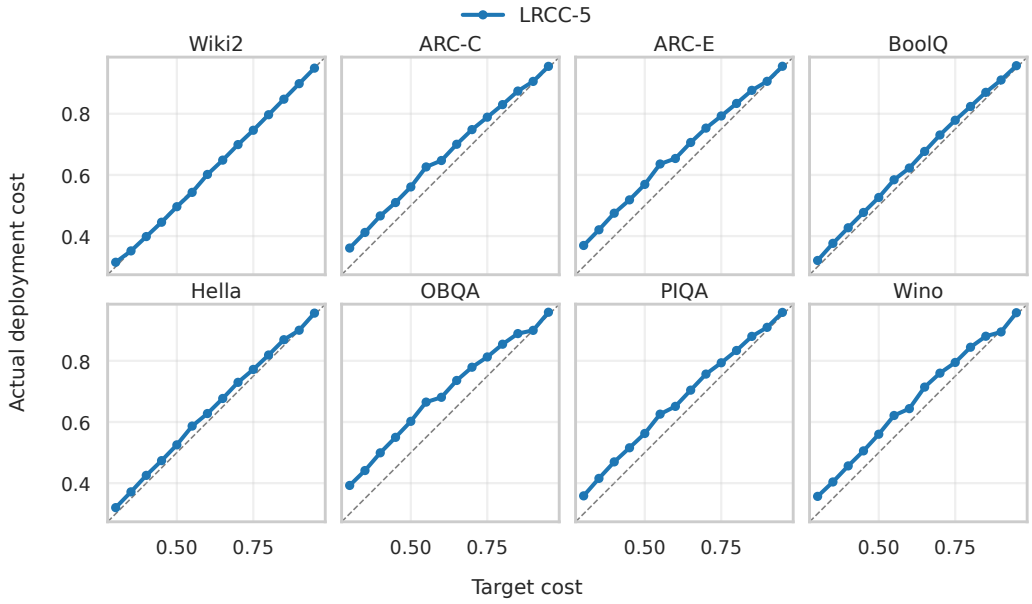


Figure 8: Target versus actual deployment cost for Llama-3.2-1B with LRCC-5. Each panel shows one evaluation task; the dashed line indicates exact agreement between the target and actual deployment cost.

a path with target cost 0.55. The results for Llama-2-7B are shown in Figure 4. Figure 10 shows the perplexities for Llama-3.2-1B. Perplexity increases sharply with only two paths, but increasing the number of choices beyond five provides comparatively little improvement. This agrees with the results for Llama-2-7B in Section 4.1.

We also evaluate the effect of the number of paths on compilation, prefill with 1,024 tokens, and decoding for models with a target cost of 0.7. For this ablation, we decode 64 tokens after prefilling with samples from WikiText-2, rather than using the complete task sequences timed in Section D.2.

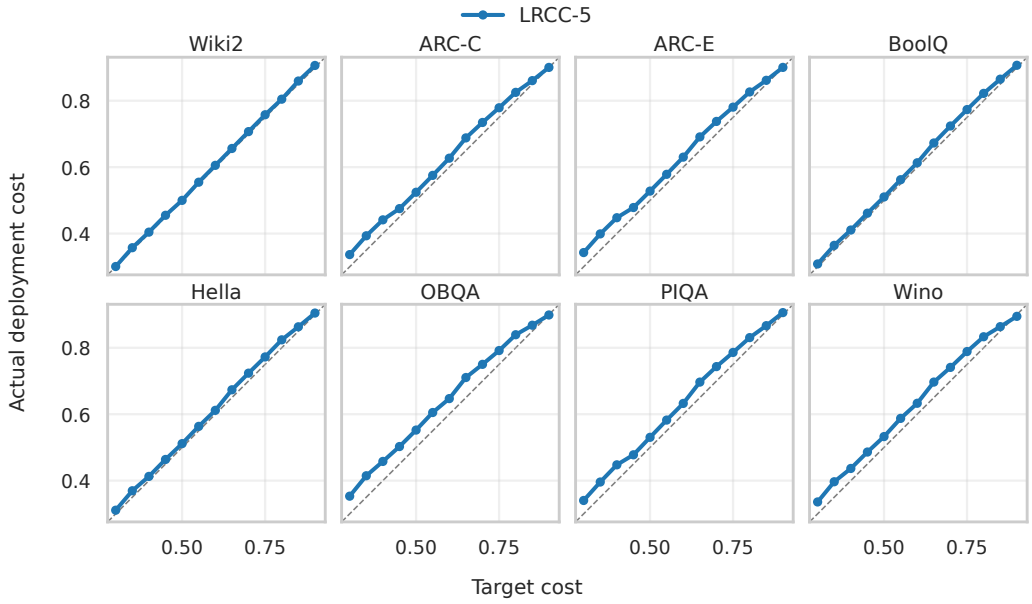


Figure 9: Target versus actual deployment cost for Qwen3-4B with LRCC-5. Each panel shows one evaluation task; the dashed line indicates exact agreement between the target and actual deployment cost.

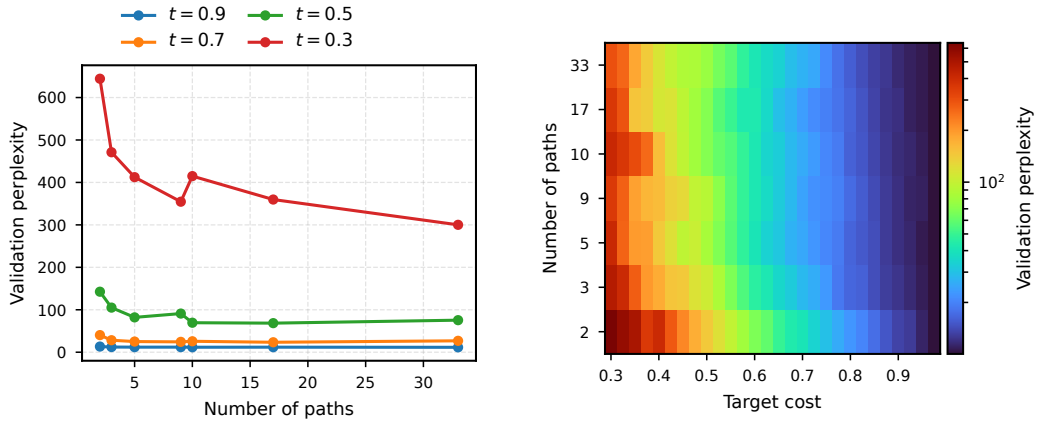
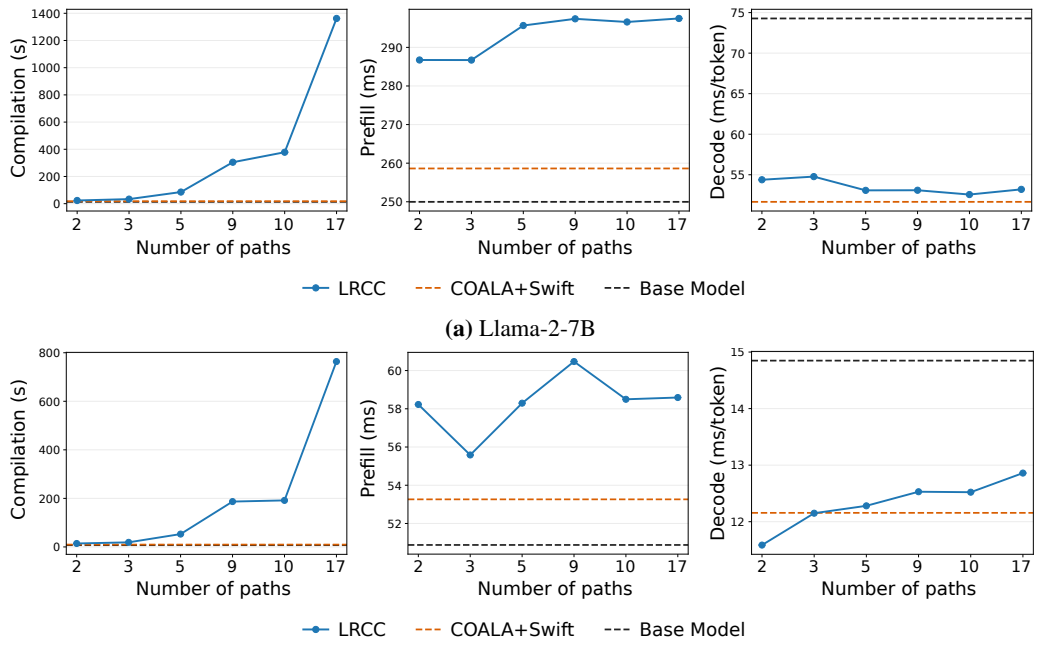


Figure 10: Ablation on the number of paths for Llama-3.2-1B. (a) shows WikiText-2 validation perplexity as a function of the number of paths for selected target costs, while (b) shows the results for all target costs from 0.3 to 0.975 in steps of 0.025.

Figure 11 shows that compilation time is the metric most affected by the number of paths. Compiling the 33-path models was infeasible because graph capture exceeded TorchDynamo's recursion depth. Compilation also became substantially slower as the number of paths increased. Increasing the number of paths has little effect on the decoding and prefill latencies of Llama-2-7B but increases the decoding latency of Llama-3.2-1B. Overall, the 5-path models provide a good trade-off between predictive and computational performance. Finally, we note that the prefill implementation of LRCC computes the first factor multiplication at full rank, masks the intermediate representation according to the selected path, and then applies the second factor, thus performing computation for ranks that are subsequently masked. The prefill implementation therefore does not yet exploit all of the computation savings available from routing; we leave its optimization to future work. Other optimizations could improve the prefill performance of both static low-rank and LRCC models relative to the dense base model, for example by fusing operations that share the same inputs (Zhu et al., 2026).



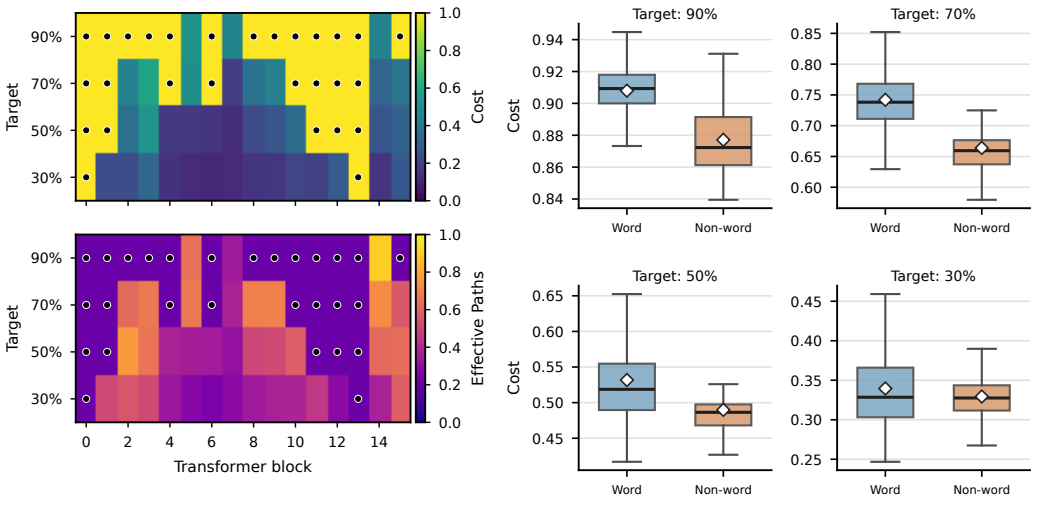
(a) Llama-2-7B

(b) Llama-3.2-1B

Figure 11: Effect of the number of paths on compilation, prefill, and decoding times. Increasing the number of paths leads to substantially higher compilation times and slightly higher decoding times.

D.5 Router Choices

We repeated the same experiments described in Section 4.2 using Llama-3.2-1B. The results are shown in Figure 12. As in Llama-2-7B, a backbone of blocks is retained in dense form, while other Transformer blocks use multiple paths across tokens. Figure 12(b) also suggests that Non-Word tokens are assigned less compute on average.



(a) **(b)**

Figure 12: Router choices for Llama-3.2-1B on WikiText-2. (a) Average per-block cost (top) and effective fraction of paths (bottom) for each target. The dots indicate blocks that are deployed without factorization. (b) Distributions of the costs assigned to word and non-word tokens for each target.

E Implementation Details

All training was conducted on A100 GPUs with 64GB of VRAM. We trained linear block-level routers on 256 WikiText-2 sequences of length 2,048 with BF16 parameters using Adam with a constant learning rate of 10^{-3} and effective batch size of 4. Training annealed over 28 target costs, from 0.975 to 0.300 in steps of -0.025 with 12 epochs per target by default and up to 30 epochs when no valid checkpoint was found. A training epoch took, on average, 2 minutes 24 seconds for Llama-2-7B and 28 seconds for Llama-3.2-1B. Both models required approximately 350 epochs to produce all 28 checkpoints, corresponding to approximately 14 hours for Llama-2-7B and 2.3 hours for Llama-3.2-1B, or about 30 and 5 minutes per checkpoint, respectively.

For the compression loss, we set the soft-cap slope to $\kappa = 20$. We initialized the compression-loss weight at $\lambda_0 = 1$. We selected the dual learning rate α for λ and the Gumbel-Softmax temperature schedule using a model-specific grid of four configurations. For Llama-3.2-1B, we searched dual learning rates $\alpha \in \{0.4, 0.8\}$ and temperature schedules of a fixed $\tau = 0.5$ and $2.0 \to 0.5$. For Llama-2-7B, we searched $\alpha \in \{1.2, 1.6\}$ and temperature schedules of a fixed $\tau = 2.0$ and $2.0 \to 0.5$. Among the four runs, we selected the configuration with the lowest WikiText-2 validation loss at the largest number of the 28 target costs. We repeated this search independently for each path count in $\{2, 3, 5, 9, 10, 17, 33\}$.

Using the authors' source code, we reimplemented COALA, Swift-SVD, and SVD-LLM in our framework to ensure consistent experimental settings and evaluation. We omit SVD-LLM's optional fine-tuning stage so that no method fine-tunes the pretrained weights after compression; LRCC additionally trains its routers. An analogous weight-refinement stage could be applied to LRCC and the baselines. For COALA, we searched accumulation values $\{2, 4\}$ and regularization values $\mu \in \{0, 1\}$, at each target cost in $\{0.3, 0.5, 0.7, 0.9\}$. We found that an accumulation value of 2 and $\mu = 0$ gave the best WikiText-2 validation performance, although differences were small. We used these values for all COALA decompositions, including the base factorization of the Llama models in LRCC. For Swift-SVD, we used their recommended layer importance weight of 0.6 in Llama-2-7B. For Llama-3.2-1B, we observed that the best results were obtained with a layer importance weight of 1.0. We used the recommended defaults for the remaining hyperparameters. SVD-LLM required no hyperparameter tuning. Finally, COALA+Swift and SVD-LLM+Swift combine the respective decompositions with the rank mappings produced by Swift-SVD. We refer to the original papers for definitions of the baseline-specific hyperparameters.

E.1 Efficient Computation

The optimal factors require only the leading right singular vectors of the layer outputs $\mathbf{Y} = \mathbf{X}\mathbf{W}$, and different methods compute this subspace in different ways. Swift-SVD (Qi et al., 2026), for example, avoids storing the full output matrix by incrementally accumulating its Gram matrix over calibration batches,

$$\mathbf{G}_Y = \mathbf{Y}^\top \mathbf{Y} = \sum_j \mathbf{Y}_j^\top \mathbf{Y}_j, \qquad \mathbf{Y}_j = \mathbf{X}_j \mathbf{W}. \tag{32}$$

Since

$$\mathbf{G}_Y = \mathbf{V}\mathbf{\Sigma}^2\mathbf{V}^\top, \tag{33}$$

a single eigendecomposition of the accumulated matrix directly provides $\mathbf{V}$ and the squared singular values.

FlexRank (Zaccone et al., 2026) uses a closely related strategy, but instead incrementally accumulates the second moment of the *input* activations,

$$\mathbf{G}_X = \mathbf{X}^\top \mathbf{X} = \sum_j \mathbf{X}_j^\top \mathbf{X}_j. \tag{34}$$

Writing $\mathbf{S} = \mathbf{G}_X^{1/2}$, the right singular vectors of $\mathbf{S}\mathbf{W}$ are the same as those of the complete output matrix because

$$(\mathbf{S}\mathbf{W})^\top (\mathbf{S}\mathbf{W}) = \mathbf{W}^\top \mathbf{X}^\top \mathbf{X}\mathbf{W} = \mathbf{Y}^\top \mathbf{Y}. \tag{35}$$

This makes the memory required for calibration independent of the number of calibration tokens.

COALA (Parkina & Rakhuba, 2025) instead proposes avoiding the explicit formation of a Gram matrix. In our matrix convention, let

$$\mathbf{X} = \mathbf{Q}\mathbf{R} \tag{36}$$

be a thin QR decomposition of the calibration activations, computed with tall-skinny QR (TSQR) when $\mathbf{X}$ is too large to materialize at once. Since $\mathbf{Q}^\top \mathbf{Q} = \mathbf{I}$,

$$\left\| \mathbf{X}(\mathbf{W} - \widetilde{\mathbf{W}}) \right\|_F = \left\| \mathbf{R}(\mathbf{W} - \widetilde{\mathbf{W}}) \right\|_F . \tag{37}$$

Furthermore,

$$(\mathbf{R}\mathbf{W})^\top (\mathbf{R}\mathbf{W}) = \mathbf{W}^\top \mathbf{R}^\top \mathbf{R}\mathbf{W} = \mathbf{W}^\top \mathbf{X}^\top \mathbf{X}\mathbf{W} = \mathbf{Y}^\top \mathbf{Y}. \tag{38}$$

Therefore, an SVD of the substantially smaller matrix $\mathbf{R}\mathbf{W}$ yields the same right singular subspace $\mathbf{V}$ in exact arithmetic. Unlike approaches based on the normal equations, this procedure does not explicitly form $\mathbf{X}^\top \mathbf{X}$ or require its inversion. This is numerically advantageous for ill-conditioned calibration activations, since forming a Gram matrix squares the condition number and can amplify floating-point errors.

In our experiments, we used COALA's QR strategy to compute the decompositions for the Llama models. For Qwen3-4B, the COALA decomposition did not converge, so we accumulated the output Gram matrix over calibration batches following Qi et al. (2026).