REASONALLOC: HIERARCHICAL DECODING-TIME KV CACHE BUDGET ALLOCATION FOR REASONING MODELS

Anonymous authors
Paper under double-blind review

ABSTRACT

Long reasoning sequences make the key-value (KV) cache a major source of inference memory use and, consequently, a critical inference bottleneck. Existing decoding-time compression methods typically propose to select tokens under uniform budgets, overlooking differences in demand across layers and KV heads. Non-uniform methods primarily target prefill compression and are not adequately tailored to the growing cache demands of reasoning models during decoding. Therefore, how to allocate KV-cache budgets across layers and KV heads according to their decoding-time demand remains largely unexplored. To bridge this gap, we propose `ReasonAlloc`, a training-free method for hierarchical KV-cache budget allocation during decoding. `ReasonAlloc` is based on our systematic attention analysis across five models and four datasets, which shows that layer-level cache demand varies across models but is similar across datasets for a given model. Within a layer, demand is concentrated in a subset of query heads. These findings motivate calibrating layer budgets once per model and adapting KV-head budgets to demand during decoding using current token scores. We decouple allocation from token selection, using current token scores to guide head budgets while retaining existing eviction rules. Experiments on mathematical reasoning and code generation show consistent gains over R-KV at the same cache budget. On DeepSeek-R1-Distill-Llama-8B, `ReasonAlloc` improves MATH-500 accuracy by 4.55-6.45 percentage points at average per-head budgets of 256, 512, and 1,024 tokens. Its throughput remains within 0.66% of R-KV at matched batch sizes and budgets. At 16K generation, cache compression enables larger batches and up to $5.52\times$ the throughput of an uncompressed cache.

1 INTRODUCTION

Memory-efficient decoding is important for deploying reasoning models on hardware with limited GPU memory. These models solve difficult problems by generating intermediate steps before a final answer (Wei et al., 2022; Guo et al., 2025). A solution can extend to thousands of tokens, each adding key and value states to the key-value (KV) cache during decoding. The growing cache leaves less memory for batching and can constrain inference throughput (Zhang et al., 2023; Cai et al., 2025).

Existing decoding-time KV-cache compression methods address this memory cost by retaining only part of the accumulated history. They periodically score cached entries and remove those that do not fit within the retained budget. Attention weights provide a common estimate of which entries remain relevant (Zhang et al., 2023). R-KV further accounts for repetition in long reasoning by combining attention importance with key-state redundancy (Cai et al., 2025). It selects entries under the same budget for every layer and KV head. Such uniform budgets do not account for differences in demand. A head that needs many earlier tokens can lose useful history even when those tokens receive high scores, while another head retains lower-scoring entries because its capacity cannot be reassigned. Better token scoring alone cannot resolve this mismatch.

Non-uniform allocation adjusts how capacity is distributed across layers and heads. For prefill compression, PyramidKV uses layer depth and DynamicKV uses input attention to set layer budgets (Cai et al., 2024; Zhou et al., 2025). Ada-KV allocates capacity among heads according to their attention

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.

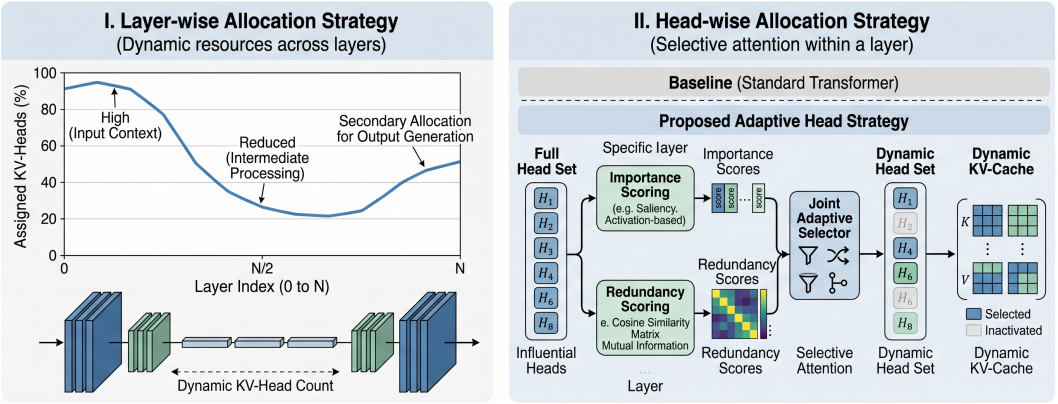

Figure 1: Overview of ReasonAlloc. Offline calibration assigns model-specific layer budgets (left). Online routing distributes each layer's budget across KV heads from current token scores (right). The diagram depicts the importance-redundancy score used in our main experiments.

scores (Feng et al., 2025). These methods address the prompt's KV cache. During decoding, however, newly generated tokens continually add to the history that may be needed for subsequent reasoning. Demand measured on the prompt may not reflect this growing history. Extending non-uniform allocation to decoding therefore requires estimating demand beyond the initial input while keeping the cost of repeated compression low.

In this paper, we examine decoding-time cache compression from the perspective of budget allocation. Our analysis asks how demand differs across layers and heads, and what these differences imply for dividing a fixed cache budget. We estimate each query head's demand by counting the smallest set of historical tokens that covers a fixed fraction of its attention mass. This count measures how broadly each head distributes attention over the available history. Across five models and four datasets, we find that the relative distribution of demand across layers differs between models but is similar across datasets within each model. Cross-dataset similarity supports reusing a layer profile for each model. Within layers, the quarter of query heads with the largest demand accounts for 45.44-56.63% of measured demand, compared with 25% under a uniform distribution. This concentration indicates that uniform head budgets give equal capacity to heads with substantially different measured demands.

Based on these observations, we propose ReasonAlloc, a training-free method for hierarchical KV-cache budget allocation during decoding (Figure 1). We combine a model-specific division of capacity across layers with flexible sharing among the KV heads within each layer. The layer budgets reflect the demand profile of the model and are calibrated once for reuse across tasks. This keeps layer profiling outside the decoding loop. Within each layer, we allocate capacity according to the current candidate scores, so that heads with more high-scoring entries can retain more history. Reusing these scores limits the additional computation needed for online head allocation. This division lets us account for demand at both levels while limiting the work added to each compression step. We keep allocation independent of the choice of token-scoring rule, allowing different compression methods to use the same allocation framework while preserving their own criteria for selecting tokens.

We evaluate ReasonAlloc on four models across MATH-500, AIME 2024, and LCB-200, a fixed subset of 200 easy LiveCodeBench problems. Our contributions are summarized as follows:

- We characterize cache demand across layers and heads through attention analysis on five models and four datasets. The analysis identifies model-specific layer profiles that are similar across the evaluated datasets, together with demand concentrated among heads within each layer. These findings provide an empirical basis for allocating capacity at both levels.
- We introduce a training-free allocator that combines layer budgets calibrated once per model with KV-head budgets updated during decoding. We separate these allocation decisions from token scoring, allowing us to redistribute capacity using existing eviction rules.
- We show that allocation improves compression accuracy under a fixed token scorer, retained budget, and refresh interval. Gains over R-KV are obtained across the evaluated models and tasks, with

comparable throughput on R1-Llama-8B at matched batch sizes and budgets. Ablations support both allocation stages, and accuracy also improves with SnapKV's attention-only score (Li et al., 2024).

With `ReasonAlloc`, we improve reasoning accuracy under a fixed cache budget, providing an allocation framework that complements token selection for memory-efficient inference.

2 Related Work

Cache eviction reduces memory by retaining selected KV entries. StreamingLLM keeps attention sinks and a recent window (Xiao et al., 2023), while H_2O combines recent tokens with those receiving high accumulated attention (Zhang et al., 2023). SnapKV selects prompt positions from attention in an observation window before generation (Li et al., 2024). R-KV addresses long reasoning outputs by combining attention importance with key-state redundancy during decoding (Cai et al., 2025). These methods focus on the rule used to select entries within a cache budget.

Non-uniform allocation adjusts how this budget is distributed. PyramidKV assigns more capacity to lower layers (Cai et al., 2024), while DynamicKV uses input-dependent attention to adjust layer budgets during prefill (Zhou et al., 2025). Ada-KV allocates capacity across heads from their attention scores (Feng et al., 2025). These approaches allocate capacity when compressing the prompt's KV cache. In `ReasonAlloc`, we combine model-specific layer calibration with head budgets updated during decoding. We compare these allocation choices under the same retained budget, token scorer, and refresh interval.

Cache memory can also be reduced by changing how stored states are represented. Quantization lowers the numerical precision of cached states, as in KVQuant (Hooper et al., 2024). MiniCache merges similar states across adjacent layers (Liu et al., 2024), while Palu stores low-dimensional representations and reconstructs keys and values when needed (Chang et al., 2024). We address the distribution of cache capacity, assigning retained-entry budgets across layers and heads under a fixed storage format.

Prompt compression reduces the input through semantic compression (Fei et al., 2024) or token selection using evaluator heads (Fei et al., 2026). Training with length constraints or mixtures of long and short reasoning traces reduces output length (Hou et al., 2025; Yu et al., 2025). These approaches change the sequences the model processes. We retain the original model and generation protocol and allocate cache capacity during decoding.

3 Preliminaries

We consider autoregressive decoding with L layers and H physical KV heads per layer. Each generated token adds one key-value pair per head. In grouped-query attention, query heads sharing a KV head reuse the same stored states.

Let $B^{(\ell,i)}$ denote the retained-entry budget for head i in layer ℓ. For a reported budget b, the global retained budget is $B = LHb$, measured in KV entries across all heads and layers. Uniform allocation sets $B^{(\ell,i)} = b$, while unequal allocations satisfy $\sum_{\ell=1}^{L}\sum_{i=1}^{H} B^{(\ell,i)} \le B$.

The cache is checked every Δ generated tokens. When compression is needed at step t, an eviction policy assigns a score $S_{j,t}^{(\ell,i)}$ to candidate token j. Candidates comprise retained history and newly added entries. Each head retains the highest-scoring candidates up to its budget. Entries accumulated between refreshes can raise peak memory above the retained budget (see Appendix J for more details).

Our main comparisons use R-KV's score (Cai et al., 2025):

$$S_{j,t}^{(\ell,i)} = \alpha I_{j,t}^{(\ell,i)} - (1-\alpha) R_{j,t}^{(\ell,i)}, \tag{1}$$

where I denotes attention importance, R denotes key-state redundancy, and α controls their relative weight. For grouped-query attention, we follow R-KV by max-pooling logits from query heads sharing a KV head before normalization as shown in Appendix C. Furthermore, we evaluate our allocation with SnapKV's attention-only score in Appendix E.

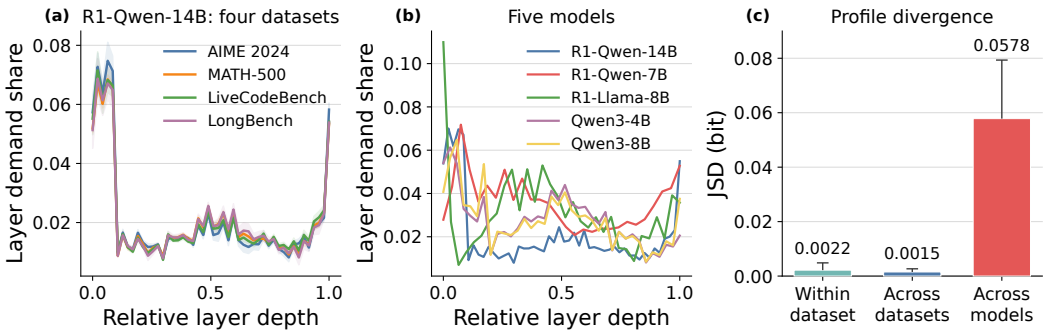

Figure 2: Normalized layer-demand profiles at $\rho = 0.93$. **(a)** Mean profiles of R1-Qwen-14B on four datasets, with shading showing the standard deviation across samples. **(b)** Sample-weighted mean profiles of five models over the same datasets. **(c)** Mean JSD on 20 relative-depth bins, using base-2 logarithms, between samples within each model-dataset pair, dataset means within each model, and model means. Whiskers extend one standard deviation above the means.

4 KV Demand Across Layers and Heads

We use attention concentration to estimate cache demand across layers and heads. For each query head and query position, we rank historical tokens by attention weight and count the smallest set whose cumulative mass reaches $\rho = 0.93$. This count serves as a proxy for cache demand. We average it over query positions to obtain head demand, then sum over query heads to obtain layer demand. The threshold is selected on a separate GSM8K development set as shown in Appendix F.

We profile R1-Llama-8B, DeepSeek-R1-Distill-Qwen-7B (R1-Qwen-7B), R1-Qwen-14B, Qwen3-4B, and Qwen3-8B on AIME 2024, MATH-500, LiveCodeBench, and LongBench. Each model-dataset pair contains ten profiles, with a generation limit of 8,192 tokens, as shown in Appendix G.

4.1 Layer profiles across datasets

The purpose of layer calibration is to capture a model's relative demand without fitting the allocation to a particular evaluation task. We therefore examine whether differences across datasets are smaller than differences across models. To compare models of different depths, we aggregate demand into 20 bins of relative layer depth and normalize each profile to sum to one. We measure differences using Jensen-Shannon divergence (JSD, in bits).

Figure 2 shows that layer profiles differ more across models than across datasets. Mean JSD is 0.0015 bit between dataset means within a model and 0.0022 bit between samples from the same dataset. Between model means, it reaches 0.0578 bit, or $38.5\times$ the cross-dataset value and $26.6\times$ the within-dataset value.

These differences concern where demand lies along depth. R1-Qwen-14B shows similar early peaks and an output-layer rise across datasets, as shown in Figure 2a. Moreover, across models, Figure 2b shows that R1-Llama-8B concentrates more demand at the start, while R1-Qwen-7B places a larger share in middle and late layers.

Finding 1 (Model-specific layer profiles)**.** *Across the evaluated datasets, normalized layer-demand profiles are more similar within a model than between models.*

On this basis, we calibrate a separate layer allocation for each model and reuse it across the evaluated tasks.

4.2 Head concentration within a layer

Layer calibration determines how much cache a layer receives, but leaves open how that capacity should be divided among heads. We examine whether a uniform division reflects the demand within each layer.

We measure concentration by the share of demand carried by the most demanding quarter of query heads. For each dataset, we first average head demands over samples, compute this share within each layer, and then average over layers and datasets. The share ranges from 45.44% to 56.63% across the five models, compared with 25% under a uniform distribution as shown in Appendix Table 4. It exceeds 50% in four of the five models.

Finding 2 (Layer allocation alone is insufficient)**.** *Uniform head budgets do not reflect the concentration of demand within a layer, and changing the layer's total capacity does not correct this mismatch.*

We therefore retain the calibrated layer budget as a constraint and use current candidate scores to determine how to distribute it among heads.

5 ReasonAlloc Framework

We use the findings in Section 4 to divide allocation into a model-level decision and a decision made during decoding. Layer calibration assigns capacity according to the model's demand profile. Within each layer, we use current token scores to direct capacity toward heads with more high-scoring history. This division lets us reuse the layer profile while adapting head budgets to the current cache.

5.1 Offline Layer-wise Preallocation

We estimate how much history each layer attends to using the same eight SQuAD calibration passages D for each model as in Appendix G.1. Let $a_{j,t}^{(\ell,q)}$ denote the attention weight from query head q in layer ℓ at position t to historical token j. Broader attention requires more tokens to cover a fixed mass ρ. We use the minimum such count as head demand:

$$k_{\ell,q,t}(\rho) = \min\left\{k : \textstyle\sum_{j\in \mathrm{Top}_k(a_{\cdot,t}^{(\ell,q)})} a_{j,t}^{(\ell,q)} \geq \rho\right\}. \tag{2}$$

Here, Top_k returns the indices of the k largest attention weights. Averaging over calibration query positions Q_D gives typical head demand, which we sum across query heads:

$$B_{\mathrm{raw}}^{(\ell)} = \sum_q \frac{1}{|Q_D|} \sum_{t\in Q_D} k_{\ell,q,t}(\rho). \tag{3}$$

Direct proportional allocation can leave little capacity for low-demand layers. We therefore temper demand differences before assigning budgets. For nonnegative demands $\boldsymbol{x} = (x_1, \ldots, x_n)$ with positive sum, our operator $\boldsymbol{y} = \mathcal{R}(\boldsymbol{x}, \nu, [a, b], B_{\mathrm{target}})$ divides a total capacity B_{target} among the n entries. B_{target} equals B for allocation across layers and $B^{(\ell)}$ for allocation across the heads of layer ℓ. We assign capacity in proportion to the smoothed demands and clip each allocation to $[a, b]$:

$$\hat{y}_k = \mathrm{clip}\left(B_{\mathrm{target}} \frac{x_k^\nu}{\sum_{m=1}^n x_m^\nu},\, a,\, b\right), \qquad 0 < \nu < 1. \tag{4}$$

We raise each demand x_k to the power $\nu \in (0, 1)$ to smooth the profile and reduce the influence of extreme demand values. This preserves the ordering of layer demands while making the allocation less concentrated. Clipping complements this smoothing by retaining allocation weight for low-demand layers and limiting large individual allocations. We enforce the fixed cache capacity after clipping by rescaling and rounding down, giving integer budgets with $\sum_k y_k \leq B_{\mathrm{target}}$:

$$y_k = \left\lfloor B_{\mathrm{target}} \frac{\hat{y}_k}{\sum_{m=1}^n \hat{y}_m} \right\rfloor. \tag{5}$$

For layer allocation, we use $\mathcal{R}$ with $n = L$, $\nu = \gamma$, and $[a, b] = [0.25\bar{B}, 2\bar{B}]$, where $\bar{B} = B/L$ is the uniform layer budget. We denote $\mathcal{R}$ with these settings by $\mathcal{R}_L$ and compute $\{B^{(\ell)}\}_{\ell=1}^L = \mathcal{R}_L(\boldsymbol{B}_{\mathrm{raw}})$ from the raw layer demands once per model. We reuse these budgets across tasks.

5.2 Online Head-wise Dynamic Routing

Head allocation should let a head with many high-scoring candidates use more of its layer's capacity. We estimate this demand from the scores used for token selection. For layer ℓ at step t, let $C_{i,t}$

Algorithm 1 Offline layer calibration and online KV-head routing in `ReasonAlloc`.

Require: Model M, calibration set D, retained budget B, attention fraction ρ, refresh interval Δ, head floor μ

for $\ell = 1$ **to** L **do**
  Estimate $B_{\text{raw}}^{(\ell)}$ on D using Eq. (3)
end for
$\{B^{(\ell)}\}_{\ell=1}^{L} \leftarrow \mathcal{R}_L(\boldsymbol{B}_{\text{raw}})$
while generation has not finished **do**
  Generate token t and append its KV entries to the candidate cache
  if $t \bmod \Delta = 0$ **then**
    for $\ell = 1$ **to** L **do**
      if $\sum_{i=1}^{H} |C_{i,t}| \le B^{(\ell)}$ **then**
        Retain all candidate entries in layer ℓ without compression
      else
        Score candidates and pool $S_{j,t}^{(\ell,i)}$ over KV heads i
        $\tau \leftarrow \textsc{KthLargest}(\mathcal{S}_t^{(\ell)}, B^{(\ell)})$
        $\hat{r}_i \leftarrow \sum_{j \in C_{i,t}} \mathbb{I}(S_{j,t}^{(\ell,i)} \ge \tau)$ for each head i
        $\{B^{(\ell,i)}\}_{i=1}^{H} \leftarrow \mathcal{R}_H^{(\ell)}(\hat{\boldsymbol{r}})$
        In each head i, retain up to $B^{(\ell,i)}$ highest-scoring tokens
      end if
    end for
  end if
end while

contain the retained history and newly accumulated entries of KV head i. We score these candidates with $S_{j,t}^{(\ell,i)}$ as defined in Section 3.

We assess each head by its contribution to the layer's highest-scoring candidates. At a refresh, we compress the layer when $\sum_i |C_{i,t}| > B^{(\ell)}$. We pool one score per candidate entry from its H KV heads into a multiset $\mathcal{S}_t^{(\ell)}$. The $B^{(\ell)}$-th largest score defines a common threshold:

$$\tau = \textsc{KthLargest}\big(\mathcal{S}_t^{(\ell)}, B^{(\ell)}\big). \tag{6}$$

For each head, we count the candidates meeting this threshold, $\hat{r}_i = \sum_{j \in C_{i,t}} \mathbb{I}\big(S_{j,t}^{(\ell,i)} \ge \tau\big)$. Sharing one threshold lets heads compete for the layer's capacity through their current token scores. Counting the candidates connects this comparison to storage demand, since each retained entry occupies one cache slot in its head. Heads with similar peak scores can thus have different demand estimates when they contain different numbers of candidates above the threshold.

We moderate concentration in the head budgets with the same allocation operator used for layers. The head smoothing exponent β controls ratios among positive smoothed demands. Smaller values bring these estimates closer together, tempering the influence of heads with many high-scoring candidates. The intermediate floor $B_{\min}^{(\ell)} = \mu B^{(\ell)}/H$ raises small allocations before renormalization. A positive floor also gives weight to heads whose candidates all fall below the shared threshold. The coefficient $\mu \in [0, 1]$ sets this floor relative to the uniform per-head budget. The head operator is

$$\mathcal{R}_H^{(\ell)}(\boldsymbol{x}) = \mathcal{R}\big(\boldsymbol{x}, \beta, [B_{\min}^{(\ell)}, B^{(\ell)}], B^{(\ell)}\big). \tag{7}$$

The resulting budgets $\{B^{(\ell,i)}\}_{i=1}^{H} = \mathcal{R}_H^{(\ell)}(\hat{\boldsymbol{r}})$ sum to at most $B^{(\ell)}$. Each head retains its highest-scoring candidates up to its assigned capacity.

At each compression step, the threshold reflects the retained history together with newly generated entries. Their changing score distribution can alter the head budgets within the fixed layer capacity. Algorithm 1 specifies the calibration and decoding procedure, and Appendix A provides a flowchart.

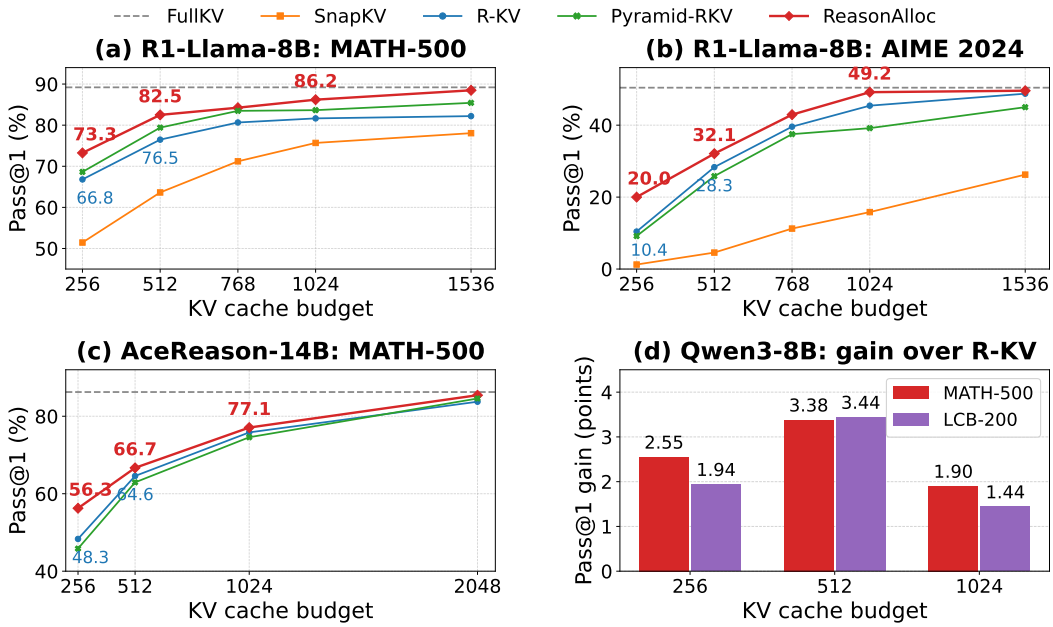

Figure 3: Pass@1 at matched retained KV budgets. Panels (a)-(c) compare accuracy on the indicated model-task pairs, with dashed lines showing FullKV. Panel (d) reports Qwen3-8B gains over R-KV in percentage points. LCB-200 is the fixed 200-problem easy subset of LiveCodeBench. Appendix Tables 5 and 6 give the numerical results.

6 Experiments

We evaluate whether reallocating a fixed cache budget improves accuracy while preserving decoding throughput. Our controlled comparisons hold the token scorer fixed to measure the effect of allocation.

6.1 Experimental Setup

We evaluate R1-Llama-8B, R1-Qwen-14B, and AceReason-14B on MATH-500 and AIME 2024. Qwen3-8B extends the study to non-thinking generation on MATH-500 and LCB-200, a fixed set of 200 easy LiveCodeBench problems. We report pass@1 from eight stochastic samples per problem using public checkpoints without further training. Appendix K specifies the checkpoints, sampling settings, generation limits, and construction of LCB-200.

We compare allocation rules using a common R-KV scorer (Cai et al., 2025), with R-KV as the uniform-budget baseline. We adapt the allocation rules of PyramidKV, DynamicKV, and Ada-KV (Cai et al., 2024; Zhou et al., 2025; Feng et al., 2025) for decoding-time compression, producing Pyramid-RKV, Dynamic-RKV, and Online AdaKV-RKV, respectively. All three adaptations use the same grouped-query aggregation and refresh interval as R-KV. We evaluate Dynamic-RKV and Online AdaKV-RKV on MATH-500 using R1-Llama-8B, together with StreamingLLM (Xiao et al., 2023) and H_2O (Zhang et al., 2023). FullKV gives the uncompressed accuracy reference, and a decoding-time implementation of SnapKV (Li et al., 2024) provides an attention-only compression baseline. Appendix K specifies the adaptations.

All decoding-time methods use the retained-budget convention in Section 3 and refresh the cache every 128 generated tokens. We use 200 GSM8K training examples for parameter selection, giving $\gamma = \beta = 0.5$ and $\mu = 0.25$, and keep these values fixed across tasks. We calibrate one layer profile per model on eight SQuAD passages disjoint from the evaluation sets and reuse it across benchmarks. Appendix F reports parameter sensitivity.

6.2 Main Results

With the R-KV scorer held fixed, as shown in Figure 3 and Table 1, we improve pass@1 at every evaluated budget across all four models. On MATH-500 with R1-Llama-8B, the gains are 4.55-6.45

Table 1: R1-Qwen-14B pass@1 accuracy (%) at each retained KV budget. Dashes indicate settings that were not evaluated.

Benchmark	Method	128	256	512	768	1024	1536	2048	2560	3072
MATH-500	FullKV	92.20	92.20	92.20	92.20	92.20	92.20	92.20	–	–
	SnapKV	28.20	46.48	73.55	78.65	83.40	86.28	88.28	–	–
	R-KV	52.43	71.20	80.85	84.25	86.80	89.40	91.20	–	–
	Pyramid-RKV	39.45	63.23	77.08	84.05	86.25	88.28	89.80	–	–
	ReasonAlloc	**54.85**	**72.40**	**82.23**	**85.28**	**88.25**	**91.40**	**91.45**	–	–
AIME 2024	FullKV	65.83	65.83	65.83	65.83	65.83	65.83	65.83	65.83	65.83
	SnapKV	0.42	2.08	12.08	17.08	26.25	35.42	45.42	48.33	52.92
	R-KV	0.42	7.50	24.17	34.58	42.50	54.17	54.58	58.75	63.75
	Pyramid-RKV	0.00	4.58	19.17	29.58	40.42	50.00	56.67	62.50	60.42
	ReasonAlloc	**1.25**	**11.25**	**27.50**	**36.67**	**44.17**	**55.42**	**56.67**	**63.33**	**65.42**

Table 2: Pass@1 accuracy (%) for the two allocation stages on R1-Llama-8B, estimated from eight samples per problem. All configurations use the R-KV scorer and are compared at the same budget.

Allocation	MATH-500			AIME 2024		
	256	512	1024	256	512	1024
Uniform (R-KV)	66.80	76.48	81.65	10.42	28.33	45.42
Layer only	70.05	79.83	84.25	13.33	30.83	46.67
Head only	71.48	81.45	84.70	17.50	31.67	45.83
Layer + head (ReasonAlloc)	**73.25**	**82.50**	**86.20**	**20.00**	**32.08**	**49.17**

percentage points at budgets 256, 512, and 1,024. On LCB-200 with Qwen3-8B, we improve accuracy at all three budgets, including a gain from 79.06% to 82.50% at budget 512. The gains cover both mathematical reasoning and code generation.

The comparison among non-uniform allocation rules tests the value of the proposed hierarchy. On MATH-500 with R1-Llama-8B, we obtain higher accuracy than the decoding adaptations of PyramidKV, DynamicKV, and Ada-KV at all three budgets (Appendix Table 7). These gains indicate that our hierarchy is better suited to the evaluated decoding-time setting than the adapted layer and head allocation rules.

We use FullKV to assess how much uncompressed accuracy can be recovered at each budget. With ReasonAlloc at budget 1,024, R1-Llama-8B is within 3.00 points of FullKV on MATH-500 and 1.25 points on AIME 2024. R1-Qwen-14B approaches FullKV at larger budgets. On AIME 2024, increasing its budget from 1,024 to 3,072 raises pass@1 from 44.17% to 65.42%, close to FullKV's 65.83%.

6.3 Component Ablations

To test the contribution of each level, we compare the full allocator with layer-only and head-only variants and uniform R-KV. Layer-only keeps head budgets uniform, while head-only keeps layer budgets uniform. All use R-KV scoring on R1-Llama-8B. Table 2 reports budgets 256, 512, and 1,024, with additional budgets in Appendix H.

Each stage improves on uniform allocation in all six task-budget pairs, and combining them gives the highest accuracy throughout Table 2. The relative gain from each stage varies with the task and budget. Head routing gives the larger improvement on MATH-500 at all three budgets and on AIME 2024 at budgets 256 and 512. Layer calibration contributes more on AIME 2024 at budget 1,024, with the same ordering at several larger budgets in Appendix H. The benefit of combining them supports addressing demand both across layers and within each layer.

The head-only comparison with Online AdaKV-RKV isolates the head-allocation rule. Both use uniform layer budgets and the R-KV scorer. We gain 2.05-2.15 percentage points on MATH-500 at the three budgets, as shown in Appendix Table 7, supporting our choice of how to convert head demand into capacity.

Table 3: R1-Llama-8B throughput (tokens/s) on one RTX 4090 at the maximum sustainable batch size of each method. FullKV uses batch 4 at 8K and batch 2 at 16K. All compressed methods use batch 8. FullKV values are repeated across budget settings for comparison.

Generation length	Retained budget	Throughput (tokens/s)			
		FullKV	SnapKV	R-KV	ReasonAlloc
8K	1024	70.45	221.58	217.35	218.78
	1536	70.45	186.41	183.54	184.62
	2048	70.45	163.26	161.39	161.91
16K	1024	39.66	222.16	218.85	218.82
	1536	39.66	186.47	183.81	184.33
	2048	39.66	162.03	160.05	159.70

6.4 System Efficiency

We measure throughput to assess the cost of allocation and the batching benefit of a bounded cache. We use R1-Llama-8B on a single RTX 4090 at generation lengths of 8K and 16K, with each method's maximum sustainable batch size, as shown in Table 3. In Appendix J, we report decoding times and runtime memory details.

We obtain comparable throughput to R-KV after adding allocation. At the same batch size of 8 and the same retained budgets, the difference ranges from -0.22% to $+0.66\%$ across the six settings. We perform layer calibration before inference and reuse token scores for online head routing, improving accuracy with little change in decoding throughput.

Bounding the cache supports larger batches as generation length increases. FullKV's maximum batch falls from 4 at 8K to 2 at 16K, while we sustain batch 8 at both lengths. At 16K and budget 1,024, we obtain 218.82 tokens/s, compared with 39.66 tokens/s for FullKV, a $5.52\times$ throughput ratio. The retained budget also controls the accuracy-throughput tradeoff. At 16K, increasing it from 1,024 to 2,048 reduces throughput by 27%, while the accuracy results show gains on MATH-500 and AIME 2024 at the larger budget (See Appendix Table 5).

6.5 Compatibility with a Second Scorer

We test whether the allocation benefit extends to attention-only compression by replacing R-KV's importance-redundancy score with SnapKV's score. We retain the calibrated layer profile and allocation rules, so the new scorer supplies both the token ranking and the scores used for head allocation. On MATH-500 with R1-Llama-8B, we improve SnapKV by 3.88-6.78 percentage points at budgets 256, 512, and 1,024, with the largest gain at 512 as shown in Appendix Table 8. The gains under both scoring rules support using allocation to improve compression across different definitions of token importance.

7 Conclusion

In this paper, we introduce ReasonAlloc, a training-free method for hierarchical KV-cache budget allocation during decoding. Our demand analysis identifies layer profiles that differ between models but are similar across the evaluated datasets, together with concentrated demand within layers. These findings motivate allocation at both levels. Our design combines layer budgets calibrated once per model with KV-head budgets derived from current token scores, redistributing capacity while retaining existing eviction rules. Across four models, we improve accuracy over R-KV on mathematical reasoning and code generation at matched cache budgets. On R1-Llama-8B, decoding throughput is comparable to R-KV at matched batch sizes. Ablations support both allocation stages, and our scorer comparison also yields accuracy gains with SnapKV's attention-only score. Together, these results show that our allocation framework improves the use of a fixed cache budget and complements token selection for memory-efficient reasoning.

AI USE STATEMENT

We used generative AI tools to aid and polish the writing of the manuscript. The experimental design, model runs, and analysis code are our own work. We did not use generative AI tools to generate synthetic data, to develop theoretical models, to formulate mathematical claims, or to assist in proofs, and these tasks do not apply to this work. We reviewed and edited all AI-assisted text and take responsibility for the final content of this work.

ETHICS STATEMENT

Our study uses publicly released language models and public benchmark datasets, and involves no human subjects, no personally identifiable information, and no newly collected data. The qualitative case study quotes model-generated text only. We see no conflicts of interest and no potential for harm beyond those already associated with deploying the underlying language models.

REPRODUCIBILITY STATEMENT

We describe every step needed to reproduce the results. Appendix G specifies the profiling protocol, Appendix F reports the hyperparameter search, Appendix K lists the models, datasets, decoding parameters, calibration protocol, and the settings of the adapted baselines, and Appendix C documents the grouped-query aggregation used by the allocator. All models and datasets are public. The implementation, the calibration scripts, and the analysis code used to produce every table and figure will be released as anonymized supplementary material.

REFERENCES

Yushi Bai, Xin Lv, Jiajie Zhang, Hongchang Lyu, Jiankai Tang, Zhidian Huang, Zhengxiao Du, Xiao Liu, Aohan Zeng, Lei Hou, et al. LongBench: A bilingual, multitask benchmark for long context understanding. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 3119–3137, 2024.

Zefan Cai, Yichi Zhang, Bofei Gao, Yuliang Liu, Yucheng Li, Tianyu Liu, Keming Lu, Wayne Xiong, Yue Dong, Junjie Hu, et al. Pyramidkv: Dynamic kv cache compression based on pyramidal information funneling. *arXiv preprint arXiv:2406.02069*, 2024.

Zefan Cai, Wen Xiao, Hanshi Sun, Cheng Luo, Yikai Zhang, Ke Wan, Yucheng Li, Yeyang Zhou, Li-Wen Chang, Jiuxiang Gu, et al. R-kv: Redundancy-aware kv cache compression for training-free reasoning models acceleration. *arXiv e-prints*, pp. arXiv–2505, 2025.

Chi-Chih Chang, Wei-Cheng Lin, Chien-Yu Lin, Chong-Yan Chen, Yu-Fang Hu, Pei-Shuo Wang, Ning-Chi Huang, Luis Ceze, Mohamed S Abdelfattah, and Kai-Chiang Wu. Palu: Compressing kv-cache with low-rank projection. *arXiv preprint arXiv:2407.21118*, 2024.

Weizhi Fei, Xueyan Niu, Pingyi Zhou, Lu Hou, Bo Bai, Lei Deng, and Wei Han. Extending context window of large language models via semantic compression. In *Findings of the Association for Computational Linguistics: ACL 2024*, pp. 5169–5181, 2024.

Weizhi Fei, Xueyan Niu, Xie Guoqing, Yingqing Liu, Bo Bai, and Wei Han. Efficient prompt compression with evaluator heads for long-context transformer inference. *Advances in Neural Information Processing Systems*, 38:154566–154593, 2026.

Yuan Feng, Junlin Lv, Yukun Cao, Xike Xie, and S. Kevin Zhou. Ada-kv: Optimizing kv cache eviction by adaptive budget allocation for efficient llm inference. In *Advances in Neural Information Processing Systems*, volume 38, 2025. doi: 10.52202/085713-3773.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Peiyi Wang, Qihao Zhu, Runxin Xu, Ruoyu Zhang, Shirong Ma, Xiao Bi, et al. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*, 2025.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. Measuring mathematical problem solving with the math dataset. *arXiv preprint arXiv:2103.03874*, 2021.

Coleman Hooper, Sehoon Kim, Hiva Mohammadzadeh, Michael W Mahoney, Yakun S Shao, Kurt Keutzer, and Amir Gholami. Kvquant: Towards 10 million context length llm inference with kv cache quantization. *Advances in Neural Information Processing Systems*, 37:1270–1303, 2024.

Bairu Hou, Yang Zhang, Jiabao Ji, Yujian Liu, Kaizhi Qian, Jacob Andreas, and Shiyu Chang. Thinkprune: Pruning long chain-of-thought of llms via reinforcement learning. *arXiv preprint arXiv:2504.01296*, 2025.

Naman Jain, King Han, Alex Gu, Wen-Ding Li, Fanjia Yan, Tianjun Zhang, Sida Wang, Armando Solar-Lezama, Koushik Sen, and Ion Stoica. Livecodebench: Holistic and contamination free evaluation of large language models for code. *arXiv preprint arXiv:2403.07974*, 2024.

Yuhong Li, Yingbing Huang, Bowen Yang, Bharat Venkitesh, Acyr Locatelli, Hanchen Ye, Tianle Cai, Patrick Lewis, and Deming Chen. Snapkv: Llm knows what you are looking for before generation. *Advances in Neural Information Processing Systems*, 37:22947–22970, 2024.

Akide Liu, Jing Liu, Zizheng Pan, Yefei He, Gholamreza Haffari, and Bohan Zhuang. Minicache: Kv cache compression in depth dimension for large language models. *Advances in Neural Information Processing Systems*, 37:139997–140031, 2024.

Math-AI. Aime 2024: Problems from the american invitational mathematics examination, 2024. URL https://huggingface.co/datasets/math-ai/aime24.

Qwen Team. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025. URL https://arxiv.org/abs/2505.09388.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837, 2022.

Guangxuan Xiao, Yuandong Tian, Beidi Chen, Song Han, and Mike Lewis. Efficient streaming language models with attention sinks. *arXiv preprint arXiv:2309.17453*, 2023.

Bin Yu, Hang Yuan, Haotian Li, Xueyin Xu, Yuliang Wei, Bailing Wang, Weizhen Qi, and Kai Chen. Long-short chain-of-thought mixture supervised fine-tuning eliciting efficient reasoning in large language models. *arXiv preprint arXiv:2505.03469*, 2025.

Zhenyu Zhang, Ying Sheng, Tianyi Zhou, Tianlong Chen, Lianmin Zheng, Ruisi Cai, Zhao Song, Yuandong Tian, Christopher Ré, Clark Barrett, et al. H2o: Heavy-hitter oracle for efficient generative inference of large language models. *Advances in Neural Information Processing Systems*, 36:34661–34710, 2023.

Xiabin Zhou, Wenbin Wang, Minyan Zeng, Jiaxian Guo, Xuebo Liu, Li Shen, Min Zhang, and Liang Ding. Dynamickv: Task-aware adaptive kv cache compression for long context llms. In *Findings of the Association for Computational Linguistics: EMNLP 2025*, pp. 8042–8057, 2025. doi: 10.18653/v1/2025.findings-emnlp.426.

A FLOWCHART OF ReasonAlloc

Figure 4 depicts the offline calibration and online refresh specified in Algorithm 1.

Figure 4 provides a visual summary of the offline calibration and online dynamic routing pipelines detailed in Section 5.

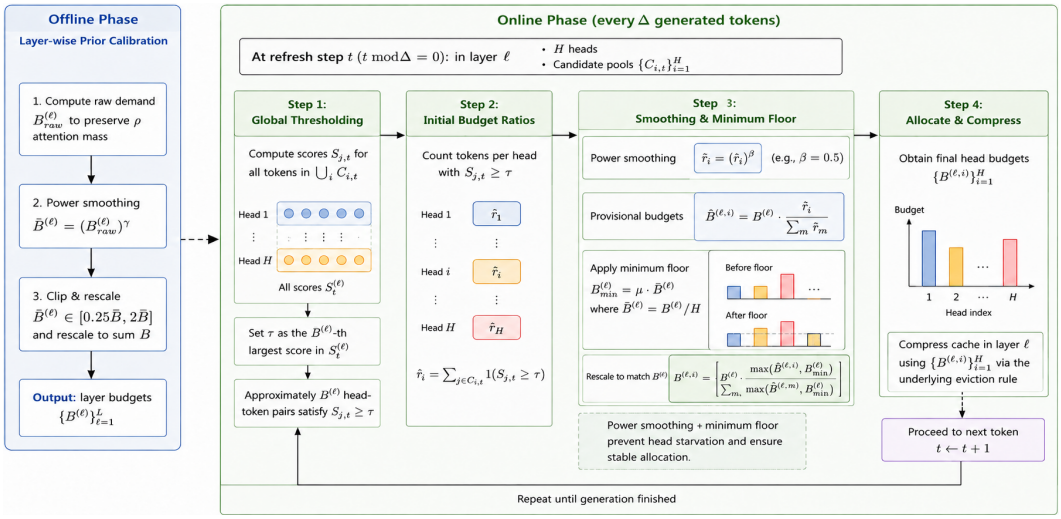

Figure 4: The ReasonAlloc pipeline. **Left (Offline):** A calibration pass measures each layer's raw KV demand at attention threshold ρ. The allocation operator in Section 5.1 maps the raw vector to per-layer budgets $\{B^{(\ell)}\}$. **Right (Online):** When a layer exceeds its budget at a Δ-step refresh, scores $S_{j,t}^{(\ell,i)}$ are pooled across its KV heads. The $B^{(\ell)}$-th largest score defines threshold τ. Per-head counts above this threshold pass through the same operator with a head-level floor to produce $\{B^{(\ell,i)}\}$. Query heads sharing one KV head are aggregated before this step.

B HEAD DEMAND ANALYSIS

Table 4 summarizes query-head demand concentration across the five models.

Table 4: Share of total query-head demand carried by the most-demanding 25% of heads and the remaining 75%. Shares are computed within each layer and averaged over layers and the four datasets: AIME 2024, MATH-500, LiveCodeBench, and LongBench. The uniform reference assigns equal demand to every head.

	Share of total demand (%)	
Model	**Top 25% of heads**	**Remaining 75% of heads**
Uniform reference	25.00	75.00
Qwen3-4B	54.54	45.46
Qwen3-8B	53.03	46.97
R1-Llama-8B	56.63	43.37
R1-Qwen-14B	51.62	48.38
R1-Qwen-7B	45.44	54.56

Figure 5 shows the query-head capacity requirements within each layer.

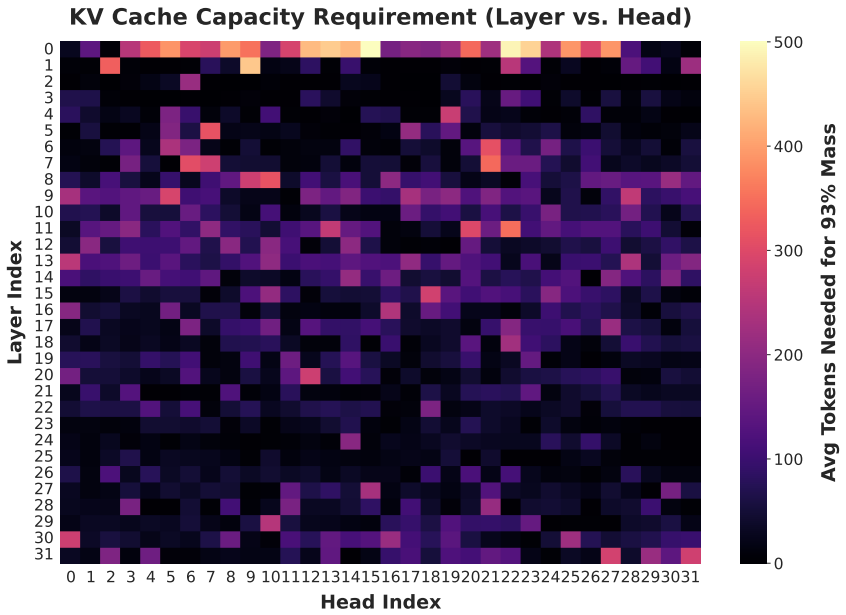

Figure 5: Query-head capacity requirements for R1-Llama-8B on MATH-500. Rows index the 32 layers and columns the 32 query heads, and each cell averages that query head's demand over the ten MATH-500 profiles. Some query heads require nearly 500 tokens to reach the target attention mass, while adjacent query heads require fewer than 50 tokens. Allocation is performed over KV heads after the GQA aggregation in Appendix C.

C GQA Head Mapping

Let H_Q and H_{KV} denote the numbers of query heads and KV heads. In grouped-query attention, each KV head is shared by $G = H_Q/H_{\mathrm{KV}}$ query heads. For KV head h, let $z_{j,t}^{(\ell,h,g)}$ be the attention logit from query head g in its group to historical token j. Following R-KV (Cai et al., 2025), we combine the group before token scoring:

$$a_{j,t}^{(\ell,h)} = \mathrm{softmax}_j \left(\max_{0 \le g < G} z_{j,t}^{(\ell,h,g)} \right). \tag{8}$$

The R-KV importance term is computed from $a_{j,t}^{(\ell,h)}$, while its redundancy term uses the key states of KV head h. The resulting score and budget are therefore defined for KV head h. Each KV state is stored once and shared by the query heads in its group.

The profile figures and head-concentration statistics report query-head attention demand because the demand metric is measured for each query head. During decoding, Equation (8) maps these signals to the KV heads that receive the budget.

D Detailed Experimental Results

Table 5 gives the full evaluated grids for R1-Llama-8B, R1-Qwen-14B, and AceReason-14B on MATH-500 and AIME 2024. Table 6 gives the exact Qwen3-8B results shown as gains in Figure 3(d). Table 7 reports the R1-Llama-8B MATH-500 baselines that were evaluated only in that setting.

Table 5: Accuracy (%) of R1-Llama-8B, R1-Qwen-14B, and AceReason-14B on MATH-500 and AIME 2024 across KV cache budgets. "-" denotes configurations that were not evaluated.

Benchmark	Model	Method	128	256	512	768	1024	1536	2048	2560	3072
MATH-500	R1-Llama-8B	FullKV	89.20	89.20	89.20	89.20	89.20	89.20	89.20	-	-
		SnapKV	33.85	51.45	63.63	71.20	75.68	78.05	79.65	-	-
		R-KV	51.83	66.80	76.48	80.65	81.65	82.20	82.83	-	-
		Pyramid-RKV	48.28	68.63	79.40	83.48	83.65	85.43	87.28	-	-
		ReasonAlloc (Ours)	**61.43**	**73.25**	**82.50**	**84.25**	**86.20**	**88.48**	**89.45**	-	-
	R1-Qwen-14B	FullKV	92.20	92.20	92.20	92.20	92.20	92.20	92.20	-	-
		SnapKV	28.20	46.48	73.55	78.65	83.40	86.28	88.28	-	-
		R-KV	52.43	71.20	80.85	84.25	86.80	89.40	91.20	-	-
		Pyramid-RKV	39.45	63.23	77.08	84.05	86.25	88.28	89.80	-	-
		ReasonAlloc (Ours)	**54.85**	**72.40**	**82.23**	**85.28**	**88.25**	**91.40**	**91.45**	-	-
	AceReason-14B	FullKV	-	86.25	86.25	-	86.25	-	86.25	-	-
		SnapKV	-	-	-	-	-	-	-	-	-
		R-KV	-	48.33	64.58	-	75.83	-	83.75	-	-
		Pyramid-RKV	-	45.83	62.93	-	74.58	-	84.58	-	-
		ReasonAlloc (Ours)	-	**56.25**	**66.68**	-	**77.08**	-	**85.43**	-	-
AIME 2024	R1-Llama-8B	FullKV	50.42	50.42	50.42	50.42	50.42	50.42	50.42	50.42	50.42
		SnapKV	0.42	1.25	4.58	11.25	15.83	26.25	32.92	39.17	41.67
		R-KV	0.42	10.42	28.33	39.58	45.42	48.75	49.17	51.67	51.25
		Pyramid-RKV	0.42	9.17	25.83	37.50	39.17	45.00	48.33	51.67	52.08
		ReasonAlloc (Ours)	**2.92**	**20.00**	**32.08**	**42.92**	**49.17**	**49.58**	**50.42**	**53.33**	**52.92**
	R1-Qwen-14B	FullKV	65.83	65.83	65.83	65.83	65.83	65.83	65.83	65.83	65.83
		SnapKV	0.42	2.08	12.08	17.08	26.25	35.42	45.42	48.33	52.92
		R-KV	0.42	7.50	24.17	34.58	42.50	54.17	54.58	58.75	63.75
		Pyramid-RKV	0.00	4.58	19.17	29.58	40.42	50.00	**56.67**	62.50	60.42
		ReasonAlloc (Ours)	**1.25**	**11.25**	**27.50**	**36.67**	**44.17**	**55.42**	**56.67**	**63.33**	**65.42**
	AceReason-14B	FullKV	-	73.33	73.33	-	73.33	-	73.33	-	-
		SnapKV	-	-	-	-	-	-	-	-	-
		R-KV	-	39.58	58.33	-	66.25	-	72.50	-	-
		Pyramid-RKV	-	34.58	58.75	-	64.17	-	71.25	-	-
		ReasonAlloc (Ours)	-	**43.75**	**61.25**	-	**68.75**	-	**73.75**	-	-

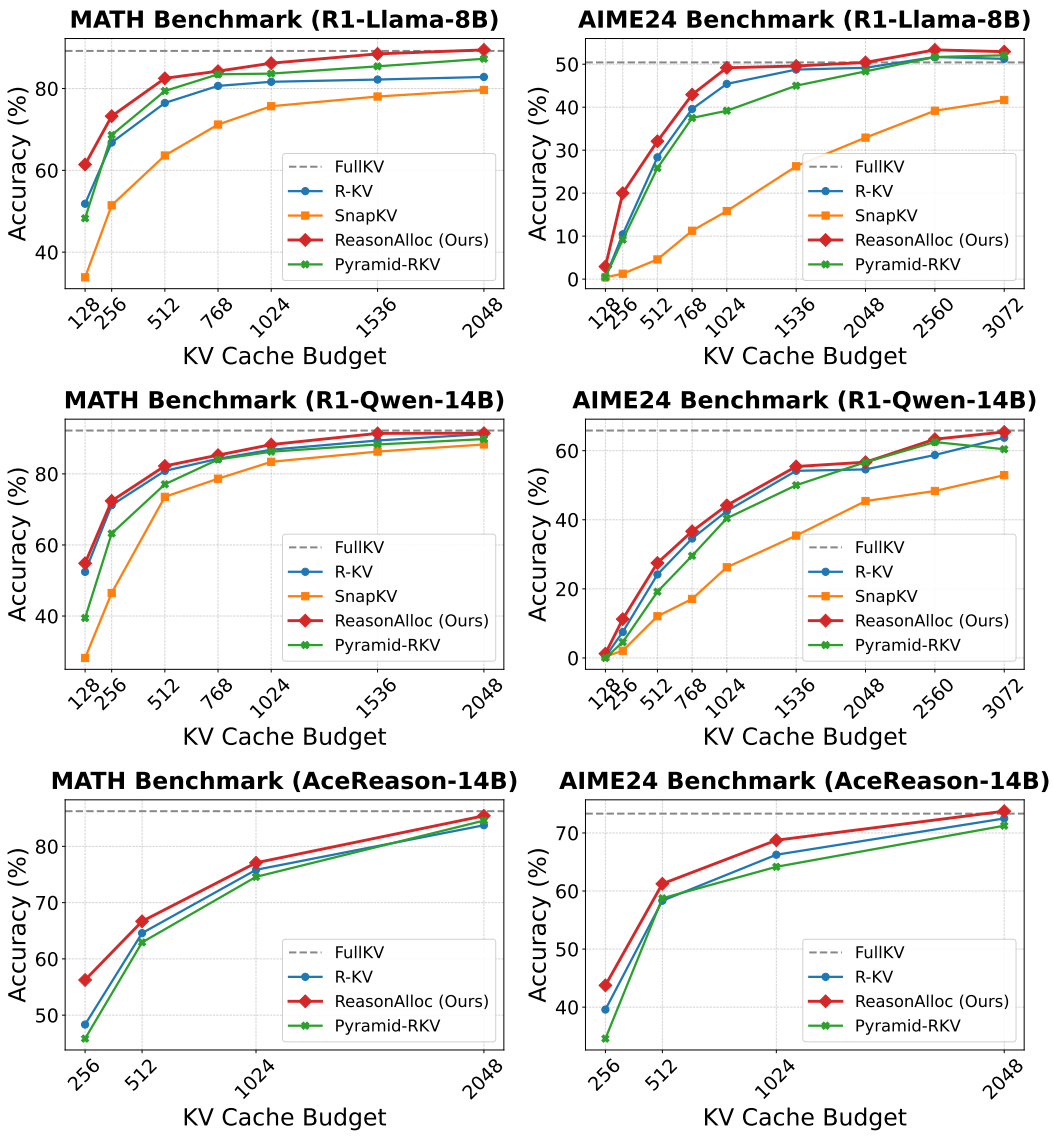

Figure 6: Accuracy on MATH-500 and AIME 2024 for R1-Llama-8B, R1-Qwen-14B, and AceReason-14B across KV cache budgets. Numerical values are reported in Table 5.

Table 6: Qwen3-8B pass@1 accuracy (%) in non-thinking mode at three retained KV budgets, with eight samples per problem. LCB-200 is the fixed easy subset of LiveCodeBench.

Benchmark	Method	256	512	1024
MATH-500	R-KV	75.83	79.25	83.80
	Pyramid-RKV	76.93	77.83	84.48
	ReasonAlloc	**78.38**	**82.63**	**85.70**
LCB-200	FullKV	82.56	82.56	82.56
	R-KV	74.56	79.06	82.06
	ReasonAlloc	**76.50**	**82.50**	**83.50**

Table 7: Additional pass@1 comparisons on R1-Llama-8B MATH-500 at three retained KV budgets, with eight samples per problem. Dynamic-RKV and Online AdaKV-RKV use the R-KV token scorer with different allocation rules. These two adapted baselines, StreamingLLM, and H_2O were evaluated in this setting only.

Method	256	512	1024
FullKV	89.20	89.20	89.20
StreamingLLM	23.43	46.88	65.05
H_2O	35.25	57.28	76.03
SnapKV	51.45	63.63	75.68
R-KV	66.80	76.48	81.65
Pyramid-RKV	68.63	79.40	83.65
Dynamic-RKV	70.43	79.05	82.88
Online AdaKV-RKV	69.43	79.38	82.55
ReasonAlloc	**73.25**	**82.50**	**86.20**

At each listed budget, ReasonAlloc exceeds the two additional adapted allocation rules.

E Compatibility with SnapKV Scoring

Table 8 gives the exact pass@1 values for the scorer-compatibility comparison in Section 6.5. Both rows use SnapKV's token score, and the calibrated layer profile is unchanged. Gains are computed before rounding the displayed accuracy values.

Table 8: Pass@1 accuracy (%) on MATH-500 with R1-Llama-8B. Both SnapKV and ReasonAlloc + SnapKV use SnapKV's attention-based token score. Absolute gain is measured in percentage points.

Method	256	512	1024
SnapKV	51.45	63.63	75.68
ReasonAlloc + SnapKV	**55.33**	**70.40**	**80.08**
Absolute gain	+3.88	+6.78	+4.40

F Hyperparameter Selection

We tune one parameter at a time using DeepSeek-R1-Distill-Llama-8B on 200 GSM8K training examples reserved as a development set. Each setting uses a 512-token budget and eight samples per problem. During each sweep, the remaining parameters are fixed to the selected values shown in Table 9.

Table 9: One-factor hyperparameter sensitivity on GSM8K dev200 (200 problems, eight samples per problem) at budget 512. The two smoothing rows vary the head-level exponent β and the layer-level exponent γ separately, each with the other held at its selected value.

Parameter	Candidates	Accuracy (%)	Selected
ρ	0.85/0.90/0.93/0.95/0.97	91.44/92.56/93.19/92.88/92.13	0.93
Head smoothing exponent β	0.25/0.50/0.75	92.13/93.19/92.06	0.50
Layer smoothing exponent γ	0.25/0.50/0.75	92.75/93.19/92.56	0.50
Floor coefficient	0.125/0.25/0.50	91.31/93.19/88.44	0.25
Refresh interval	64/128/256	91.56/93.19/90.44	128

G Profile Protocol

The profile analysis covers AIME 2024, MATH-500, LiveCodeBench, and LongBench with R1-Llama-8B, R1-Qwen-7B, R1-Qwen-14B, Qwen3-4B, and Qwen3-8B. All profile runs use temperature

0.6, top-p 0.95, and a generation limit of 8,192 tokens (8K). Each model and dataset pair contains ten profiles.

For Figure 2c, we aggregate each profile into 20 bins over relative layer depth, each spanning 5% of the model, and normalize bin demands to sum to one. For two profiles p and q, we compute $\mathrm{JSD}(p, q) = \frac{1}{2} D_{\mathrm{KL}}(p\|m) + \frac{1}{2} D_{\mathrm{KL}}(q\|m)$, where $m = (p + q)/2$ and logarithms use base 2. We compare sample pairs within each model-dataset pair, dataset means within each model, and sample-weighted model means. Figure 2c reports mean divergences in bits, with upper whiskers indicating one standard deviation across the corresponding pairs.

G.1 Calibration-Set Size

The offline layer profile is estimated once per model from a small set of calibration passages, and the size of that set is a design choice. We set it from the estimation error of the profile itself. We profile 32 SQuAD training passages for each of the five models, draw subsets of k passages, and compare the profile estimated from each subset with the profile estimated from all 32 passages. The calibration passages come from SQuAD training data and are disjoint from every evaluation set.

The estimation error falls quickly with k (Table 10). At $k = 8$, the mean 1D Wasserstein distance to the 32-passage estimate is 0.0035, or 7.9% of the cross-model distance measured with the same metric (0.0446). After the allocation operator, the largest change in a single layer's share of the total budget is 0.15 percentage points, with an average change of 0.03 percentage points over layers. Over the same models, the model-specific deviation from uniform allocation is 1.70 percentage points on average. Eight passages therefore estimate the layer budgets to well within the model-to-model differences that the allocation exploits, and doubling the set to 16 passages changes the budgets only marginally further. We therefore calibrate on the same eight passages for every model. Each model obtains one layer allocation, which is then fixed across benchmarks.

Table 10: Estimation error of the offline layer profile as a function of the number of calibration passages, averaged over the five models at $\rho = 0.93$. For each k, subsets are drawn from the 32 profiled passages and compared with the profile estimated from all 32. Fractions use the mean cross-model Wasserstein distance of 0.0446 as the reference.

Calibration passages k	**1**	**2**	**4**	**8**	**16**
Mean 1D Wasserstein distance	0.0117	0.0079	0.0054	0.0035	0.0020
Fraction of cross-model distance	26.2%	17.7%	12.0%	7.9%	4.5%

H Detailed Ablation Study Results

Table 11 reports the AIME 2024 ablation across nine budgets, Table 12 reports the MATH-500 ablation, and Figure 7 shows the AIME 2024 curves.

Table 11: Ablation study on the AIME 2024 benchmark (30 problems, eight samples per problem). We decouple `ReasonAlloc` to verify the individual contributions of the layer-wise preallocation and the online head-wise dynamic routing.

Ablation Configuration	**128**	**256**	**512**	**768**	**1024**	**1536**	**2048**	**2560**	**3072**
R-KV (Baseline, Uniform Alloc.)	0.42	10.42	28.33	39.58	45.42	48.75	49.17	51.67	51.25
+ Layer-wise Allocation Only	0.83	13.33	30.83	38.75	46.67	48.75	48.33	**55.00**	52.08
+ Head-wise Allocation Only	0.83	17.50	31.67	42.08	45.83	48.75	46.67	52.50	51.67
`ReasonAlloc` (Full)	**2.92**	**20.00**	**32.08**	**42.92**	**49.17**	**49.58**	**50.42**	53.33	**52.92**

Table 12: Ablation study on MATH-500 (500 problems, eight samples per problem, R1-Distill-Llama-8B).

Configuration	256	512	1024
R-KV (Baseline, Uniform Alloc.)	66.80	76.48	81.65
+ Layer-wise Allocation Only	70.05	79.83	84.25
+ Head-wise Allocation Only	71.48	81.45	84.70
ReasonAlloc (Full)	**73.25**	**82.50**	**86.20**

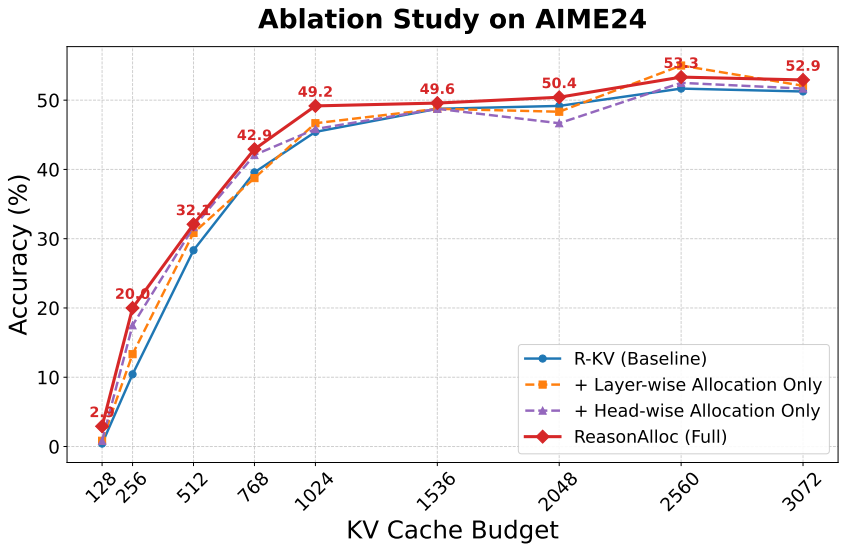

Figure 7: Ablation study of ReasonAlloc on the AIME 2024 dataset. We evaluate the individual contributions of head-wise and layer-wise allocation strategies. Combining both dimensions (ReasonAlloc Full) gives the highest accuracy at eight of the nine evaluated budgets.

I Case Study on Complex Mathematical Reasoning

We compare R-KV and ReasonAlloc on Problem 11 from the 2024 AIME II.

The problem requires finding the number of triples (a, b, c) of nonnegative integers satisfying $a + b + c = 300$ and $a^2b + a^2c + b^2a + b^2c + c^2a + c^2b = 6,000,000$. The correct answer is 601.

The baseline model correctly initiates the algebraic substitution, defining $Q = ab + bc + ca$ and $P = abc$, and arrives at the intermediate state $100Q - P = 2,000,000$. In this example the later steps drift away from the constraint introduced earlier in the derivation and move to trial substitutions:

Baseline RKV Generation Excerpt (Failed)

...
So, $100Q - P = 2,000,000$. *Hmm, that's an interesting equation. Let me write it as:* $P = 100Q - 2,000,000$.
... [Context lost] ...
Let me divide both sides by 3 to simplify: $a^3 - 300a^2 + 30,000a - 1,000,000 = 0$. *Hmm, that's a cubic equation. Maybe I can factor it or find rational roots... Let me try* $a = 100$*... Not a root. How about* $a = 120$*... Not a root...*
...
Case 1: One of the variables is zero... Case 2: All variables are equal... Thus, total solutions: $6+1 = 7$.
Final Answer: 7.

With ReasonAlloc, the trace reuses the intermediate constraint in the cubic polynomial and obtains the correct answer:

ReasonAlloc Generation Excerpt (Succeeded)

...
So, we have $P = 100Q - 2,000,000$.
We also know from Vieta's formulas that for a cubic equation with roots a, b, c: $t^3 - 300t^2 + Qt - P = 0$.
Substituting P *into the cubic equation:* $t^3 - 300t^2 + Qt - (100Q - 2,000,000) = 0$.
...
So x = 100 is a root... Therefore, the cubic factors as $(x - 100)(x^2 - 200x + (Q - 20,000)) = 0$.
... [Successfully derives the general form and permutations] ...
Therefore, the number of ordered triples is 601. Final Answer: 601.

I.1 Analysis of Reasoning Dynamics

The two traces show two different reasoning trajectories on the same problem. R-KV assigns the same budget to every layer and head. In this example its output stops using the polynomial constraint derived earlier and proceeds by case analysis, producing an incorrect count of 7.

With ReasonAlloc on the same problem, the budget is divided across layers by the calibrated demand profile and across heads by the online router of Section 5.2, which ranks candidates with the mixed utility score. In this trace the substitution $P = 100Q - 2{,}000{,}000$ and the symmetric-polynomial structure are used explicitly in later steps, and the model reaches the correct answer 601. The contrast between the two traces illustrates how a compressed cache can change the course of a long derivation.

J System Efficiency

Figure 8 compares system-level throughput across cache budgets. Table 13 reports maximum sustainable batch sizes, throughput, and total decoding times. Measurements use a single NVIDIA RTX 4090 (24 GB VRAM), as described in Appendix K.

The bounded KV cache allows ReasonAlloc to sustain larger batches as generation length grows. FullKV memory grows with sequence length, so the maximum batch size drops from 4 at 8K generation to 2 at 16K generation, yielding only 39.66 tokens/s. With a 1,024-token cache budget, ReasonAlloc sustains a batch size of 8 in both settings. At a 16K generation length, it reaches 218.82 tokens/s, a $5.52\times$ speedup over FullKV.

The hierarchical allocation itself adds negligible overhead. Head-wise routing is invoked periodically, and its operations are vectorized over heads. Across the six matched settings in Table 13, ReasonAlloc's throughput differs from R-KV by -0.22% to $+0.66\%$. At 16K generation and a 1,024-token retained budget, the rates are 218.82 and 218.85 tokens/s, respectively. Together with the accuracy results in Section 6, these measurements show that ReasonAlloc improves accuracy under compression while largely preserving R-KV's throughput.

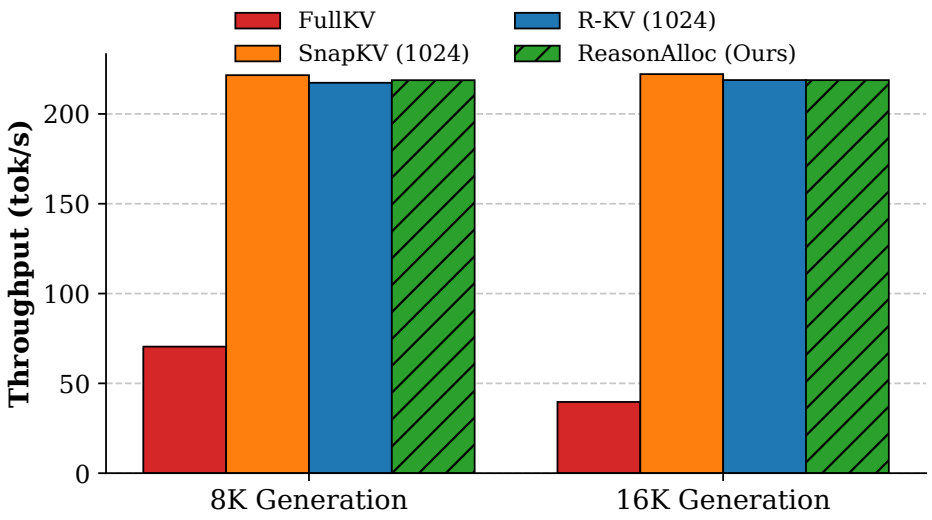

Figure 8: System-level throughput (tokens/s) for DeepSeek-R1-Distill-Llama-8B. `ReasonAlloc` achieves up to a $5.52\times$ speedup over FullKV at a 16K generation length and matches the throughput of R-KV at a 1,024-token budget.

Table 13: System-level efficiency comparison for DeepSeek-R1-Distill-Llama-8B. Compression methods support larger batches by limiting KV-cache growth. At a 16K generation length, `ReasonAlloc` reaches up to $5.52\times$ the throughput of FullKV. Its throughput is close to the other compression methods at matched cache budgets.

Gen. Length	Method	Budget	Max Batch	Throughput (tok/s)	Speedup	Dec. Time (s)
8K	FullKV	N/A	4	70.45	1.00×	465.10
	SnapKV	Fixed 1024	8	221.58	3.15×	295.77
		Fixed 1536	8	186.41	2.65×	351.57
		Fixed 2048	8	163.26	2.32×	401.41
	R-KV	Fixed 1024	8	217.35	3.09×	301.52
		Fixed 1536	8	183.54	2.61×	357.07
		Fixed 2048	8	161.39	2.29×	406.07
	ReasonAlloc	Dynamic 1024	8	218.78	3.11×	299.56
		Dynamic 1536	8	184.62	2.62×	354.97
		Dynamic 2048	8	161.91	2.30×	404.77
16K	FullKV	N/A	2	39.66	1.00×	826.27
	SnapKV	Fixed 1024	8	222.16	5.60×	589.98
		Fixed 1536	8	186.47	4.70×	702.93
		Fixed 2048	8	162.03	4.09×	808.95
	R-KV	Fixed 1024	8	218.85	5.52×	598.92
		Fixed 1536	8	183.81	4.63×	713.07
		Fixed 2048	8	160.05	4.04×	818.93
	ReasonAlloc	Dynamic 1024	8	218.82	5.52×	598.99
		Dynamic 1536	8	184.33	4.65×	711.07
		Dynamic 2048	8	159.70	4.03×	820.73

Retained budget and runtime memory. Following R-KV's distinction between retained cache and new-token buffer (Cai et al., 2025), B specifies the retained cache budget after each refresh. Algorithm 1 checks the cache every $\Delta = 128$ generated tokens and compresses a layer only when its candidate count exceeds its layer budget. For L layers with H KV heads each, and without additional per-token eviction, the buffer can add approximately $LH\Delta$ KV entries across all heads and layers before the next refresh. The retained-cache-plus-buffer component can thus approach $B + LH\Delta$, and the candidate pool before compression can exceed the retained budget. Peak memory also depends on the initial prompt, observation-window handling, temporary scoring tensors, and cache reordering. We use the same retained-cache budgets and refresh interval as R-KV and the other decoding-time allocation baselines to ensure a fair comparison of budget allocation strategies.

K Reproducibility Details

The profile study uses DeepSeek-R1-Distill-Qwen-7B, DeepSeek-R1-Distill-Llama-8B, DeepSeek-R1-Distill-Qwen-14B (Guo et al., 2025), Qwen3-4B, and Qwen3-8B (Qwen Team, 2025). Accuracy experiments use DeepSeek-R1-Distill-Llama-8B, DeepSeek-R1-Distill-Qwen-14B, AceReason-Nemotron-14B, and Qwen3-8B. We use the public checkpoints without additional training.

The datasets cover mathematical reasoning, code generation, and long-context understanding. MATH-500 (Hendrycks et al., 2021) contains 500 mathematical reasoning problems, AIME 2024 (Math-AI, 2024) contains competition problems, LiveCodeBench (Jain et al., 2024) evaluates code generation, and LongBench (Bai et al., 2024) is used for the demand profile analysis.

All adapted baselines share the R-KV token scorer, its score normalization, the grouped-query aggregation of Appendix C, and the cache-update interval $\Delta = 128$ tokens. They differ only in how the retained budget is divided. Pyramid-RKV applies PyramidKV's arithmetic layer schedule with uniform head budgets, Dynamic-RKV fixes a non-uniform layer budget derived from the input at prefill and keeps uniform head budgets during decoding, and Online AdaKV-RKV keeps uniform layer budgets and redistributes each layer's budget across KV heads online using Ada-KV's head rule, which blends the head-wise allocation with a uniform one at ratio 0.2.

Preliminary runs with tight KV budgets gave very low pass@1 on harder LiveCodeBench problems, making differences among allocation methods hard to observe. We therefore evaluate a fixed easy-labeled subset. The reported code-generation results apply to this subset. The LCB-200 subset is built from the cumulative `release_v4` pool of LiveCodeBench's `code_generation_lite`. We keep the questions that carry the release's `easy` label, de-duplicate them by question ID, and take the first 200 in the release's source order. We evaluate each generated program using all public and private test cases provided for its problem in this release. The questions span contest dates from 2023-05-07 to 2024-08-03, and the resulting subset is fixed and selected deterministically.

Qwen3-8B is evaluated with its native chat template and thinking disabled in that template, temperature 0.6, top-p 0.95 and eight samples per problem.

For mathematical reasoning tasks, we extract the answer from $\texttt{\textbackslash boxed}\{\ldots\}$, falling back to the last number in the response if no boxed answer is found. For code generation, we evaluate generated programs using the test cases associated with each problem in LCB-200.

We use observation window size 8 and importance and redundancy weight $\alpha = 0.1$. The GSM8K dev200 sweep selects $\rho = 0.93$, smoothing exponents $\gamma = \beta = 0.5$, refresh interval $\Delta = 128$, layer clipping range $[0.25\bar{B}, 2\bar{B}]$, and head floor coefficient $\mu = 0.25$. These values are fixed for the reported evaluations. Table 9 gives the measured sensitivity.

The adapted baselines keep their published allocation rules. Dynamic-RKV refreshes its cross-layer allocation every four layers during prefill and caps each layer's history budget at ten times the base budget. Online AdaKV-RKV blends the score-derived head allocation with the uniform allocation, $B^{(\ell,i)} = (1 - r)w_i + r\,(B^{(\ell)}/H)$, where the floor ratio r sets the weight of the uniform part. We checked both knobs on the same GSM8K dev200 protocol used for our own hyperparameters (200 problems, eight samples per problem, budget 512). The default floor ratio $r = 0.2$ with smoothing exponent 0.5 is the best of the evaluated settings (92.38%), ahead of $r = 0.1$ (91.81%), $r = 0.3$ (91.13%), smoothing 0.75 (91.19%) and smoothing 0.25 (91.50%), so the adapted baselines

keep their defaults. All baselines reuse the same scorer, aggregation and cache-update interval as `ReasonAlloc`.

For decoding, we sample with temperature 0.6 and top-p 0.95, drawing k=8 independent samples per query and reporting pass@1. The maximum generation length is 16,384 tokens for MATH-500 and LCB-200. For AIME 2024, it is 32,768 tokens.

Accuracy experiments in Tables 5 and 11 run on a server with 8× NVIDIA H20 GPUs. Throughput and batch-size measurements (Table 13, Figure 8) are run on a single NVIDIA RTX 4090 (24 GB VRAM) to reflect a consumer-grade single-GPU deployment scenario. We use PyTorch with the HuggingFace Transformers serving stack.

Offline layer-wise calibration uses the same fixed set of eight SQuAD passages for every model. Each model yields one layer allocation, which is then fixed across all benchmarks. For each head and query position, we record the minimum number of top-attended tokens whose cumulative attention mass reaches $\rho = 0.93$. We average these counts over query positions and sum them across heads as defined in Eq. (3). The resulting layer-demand vector passes through the allocation operator once and is cached for subsequent inference. Only the relative shape of this vector is used, so calibration only needs to estimate how demand is distributed over depth. GSM8K is used only for hyperparameter selection, and the calibration and development prompts do not overlap the evaluation prompts. Appendix G.1 reports the calibration-set size analysis used to set the number of calibration passages.

Accuracy values are pass@1 computed from $k = 8$ stochastic samples per query under the same decoding configuration for all compared methods. Throughput values are averaged over at least three generation passes.