BUDGET-AWARE EXPERT LOADING FOR MEMORY-CONSTRAINED MOE INFERENCE

Anonymous authors
Paper under double-blind review

ABSTRACT

Mixture-of-experts (MoE) models only activate few experts per token, yet CPU-offloaded inference can still incur substantial overhead from repeatedly loading missing experts to GPU when its memory is limited. In this paper, we propose Lyapunov-LRU, a training-free controller that explicitly coordinates expert loading under a fixed GPU cache capacity and a soft model-wide loading budget. The controller values each load by its marginal router-mass recovery over resident alternatives and compares this gain against a shared price updated by a request-local Lyapunov queue. Coupled with global LRU eviction, the controller coordinates loading across layers and decoding steps rather than loading every missing native-route expert. From a theoretical perspective, we prove an upper bound on the total number of loadings per request, thereby ensuring that our controller is time-efficient. We then conduct extensive experiments, involving three MoE models and four tasks spanning reasoning and code generation. The results demonstrate that Lyapunov-LRU substantially reduces expert-weight traffic and improves decode throughput, while achieving a comparable performance. For example, on MATH L5, using Lyapunov-LRU to run Qwen3-30B-A3B on L20 achieves a speedup of 1.38× and only incurs a tiny 4.4 percentage-point accuracy drop.

1 INTRODUCTION

Mixture-of-experts (MoE) large language models become popular in recent years. By activating only a small subset of experts for each token, MoE architectures expand model capacity without proportionally increasing per-token computation (Fedus et al., 2022; DeepSeek-AI et al., 2024). Qwen3-30B-A3B, for example, contains 30.5 billion parameters but activates about 3.3 billion per token (Qwen Team, 2025). Sparse activation, however, does not imply sparse storage: different tokens can require different experts throughout generation, and the model still needs to store all the parameters in memory. Therefore, limited GPU memory is still a practical obstacle to local deployment of these MoE models.

CPU offloading addresses this storage problem by keeping expert weights in host memory and transferring them to the GPU during execution. If the GPU memory is not enough to store all the parameters, a natural solution is to use a GPU expert cache, which reuses resident weights across decoding steps (Xue et al., 2024; Kamahori et al., 2025). Caching inherently involves trade-offs among space, time, and quality. For example, with a fixed and insufficient space, reducing transfers by substituting resident experts for missing ones introduces a quality–time trade-off: fewer loads can accelerate execution, but the altered route may degrade answer quality. Then an important question arises:

Under space constraints, time constraints, or both, how can we use the cache efficiently to improve model performance?

In this paper, we aim to answer the above question. We abstract the time cost as the number of admissions (i.e., loading a nonresident expert into the cache). We use B as a soft budget for average admissions per (decoded) token, aggregated across all MoE layers, and apply a Lyapunov-type algorithm to control the cache (Neely, 2010). Unlike using a hard per-token admission constraint, this controller permits bursts when routing benefits are high, allowing the number of loading to vary across layers and decoding steps, which is more efficient and robust.

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.

The key question then becomes "which cache misses are worth loading". An intuitive solution is to use router probabilities, but directly relying on them can still be misleading. For example, a missing expert can have high router probability yet add little over a resident substitute with a nearly equal probability. We therefore value an admission by the increase in retained router probability mass over the resident route, normalized by the native TopK mass.

Based on this metric, we propose our training-free controller Lyapunov-LRU. When selecting admission, the controller compares the marginal routing recovery of each admission with a shared, history-dependent loading price, raising the price after overspending and lowering it after underspending. As a theoretical guarantee, we prove an upper bound on the number of admissions for Lyapunov-LRU. When evicting experts out of the cache, the controller uses a Least Recently Used (LRU) method. Selected experts are then executed using each model's mixture-weighting rule. Note that in our model, sometimes the selected experts are different from the native routing output, which saves time cost but may lead to a performance drop.

We evaluate DeepSeek-V2-Lite-Chat, Qwen1.5-MoE-A2.7B-Chat, and Qwen3-30B-A3B on reasoning and code-generation tasks, combining logical quality–admission evaluation with real CPU–GPU offloading. The experimental results show that our controller Lyapunov-LRU can reduce the time cost significantly with only a tiny price on accuracy drop. For example, on the evaluated 160-question MATH L5 subset, using Lyapunov-LRU to run Qwen3-30B-A3B on L20 achieves a speedup of $1.38\times$ over exact-route (always loading all the experts in native routing output), with accuracy of 83.1% versus 87.5%.

Our contributions are summarized as follows:

- **Budget-aware expert loading.** We introduce a training-free controller that combines shared online pricing and global residency management under memory and loading constraints.
- **Finite-request guarantees.** Theoretically, we prove an upper bound on the cumulative admissions within each request.
- **Quality and efficiency evaluation.** We evaluate quality–efficiency trade-offs across three MoE models. The results demonstrate that Lyapunov-LRU substantially reduces expert-weight traffic and improves decode throughput, while achieving a comparable performance.

2 Problem Formulation: Residency and Loading

2.1 MoE Routing Preliminaries

A routed MoE layer consists of E experts and a router. Given an input hidden state h, the router produces logits $z \in \mathbb{R}^E$ and converts them, according to the model's native routing rule, into non-negative routing probabilities $q \in \mathbb{R}^E$. The native routing rule selects the K experts with the largest probabilities,

$$S^{\mathrm{nat}} = \mathrm{TopK}(q),$$

which we refer to as the *native route*. The selected experts process the same input independently, and their outputs are combined according to the model's native mixture-weighting rule to generate the input hidden state of the next layer.

2.2 GPU Expert Cache

Consider a fixed MoE model with L layers managed by a GPU cache: Each layer has E experts, and we represent an expert as (ℓ, e), where ℓ is the layer and e is the index. The native router selects K of them for each token. The GPU cache can hold at most LC experts, where C is the average number of cached experts per layer.

The cache is filled during initialization. Let R be the set of cached experts, and we denote $R_\ell = \{e : (\ell, e) \in R\}$ the set of cached experts in layer ℓ. Throughout decoding, we maintain

$$|R| = LC, \qquad |R_\ell| \geq K \text{ for all } \ell, \qquad K \leq C \leq E. \tag{1}$$

Note that it is a shared cache: individual layers can use more or fewer than C slots. However, since decoding one token requires at least K experts, we require $|R_\ell| \geq K$ so that it can always keep working even when no new expert is loaded.

2.3 Expert Loading and Budget

Let t index the decoding steps and let $a_{t,\ell}$ count the experts newly loaded for layer ℓ at step t. We call each new load an *admission*. Each admission replaces one cached expert: eviction requires no weight transfer back to the CPU because the original weights remain there, but loading requires to transfer weights to GPU and therefore incurs a transfer cost.

For T decoding steps, let $A_t = \sum_{\ell=0}^{L-1} a_{t,\ell}$ be the number of loads at step t and $A = \sum_{t=0}^{T-1} A_t$ their total. The copy test in Figure 1 shows an approximately linear relationship between the number of expert loads and copy time over the measured range.

We use $B \geq 0$ as a soft budget for the average number of loads per decoding step across all L layers. For example, $B = 2$ targets two new expert loads across the model per step on average. Some steps may load more experts and others fewer; our controller bounds the cumulative excess over BT within a request.

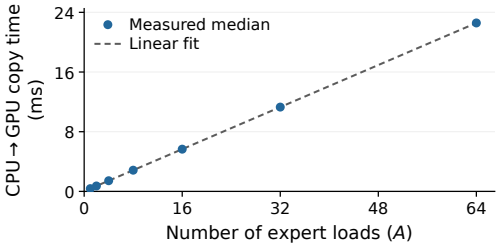

Figure 1: CPU-to-GPU copy time on A40 for expert-sized buffers of 9 MiB each. Points show median times, bars show the 10th–90th percentiles, and the line is approximately linear. Details: Section D.2.

2.4 Online Loading Decisions

Given C and B, our goal is to preserve answer quality while controlling expert loads. This requires comparing a missing expert with the alternatives already in the cache, and deciding where to spend the loading budget across layers and decoding steps. Since the final answer is not yet known, we estimate loading benefit from the router scores; Section 3 defines this estimate and the loading rule.

3 Budget-Aware Expert Loading

Under the soft loading budget B, the central question is which cache misses are worth paying for. Therefore, our controller first estimates the routing benefit of loading missing experts, and then compares this benefit with a shared loading price that reflects the loading pressure accumulated within the current request. Figure 2 illustrates the overall inference framework.

3.1 Loading Benefit from Router Scores

Router scores are available before expert execution and do not require the corresponding expert weights to be resident in GPU memory. We can therefore evaluate the potential value of a missing expert based on the router scores, before deciding whether to load it.

We now formalize this benefit. At decoding step t and layer ℓ, let $q_{t,\ell}$ denote the router probabilities over all E routed experts, and let $R_{t,\ell}$ contain the experts of that layer currently in the cache. Rank the nonresident experts by decreasing router probability, and let $I_{t,\ell}(n)$ contain the first n. If these n experts are loaded, the executable route is the TopK experts in $R_{t,\ell} \cup I_{t,\ell}$, i.e.,

$$S_{t,\ell}(n) = \mathrm{TopK}(q_{t,\ell} \mid R_{t,\ell} \cup I_{t,\ell}(n)) .$$

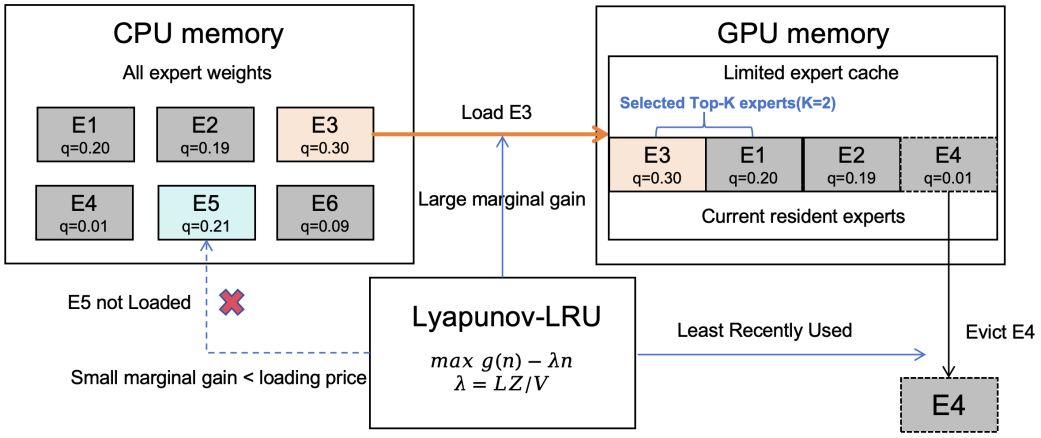

Figure 2: Overview of Lyapunov-LRU. Expert weights are stored in CPU memory, while the GPU maintains a limited expert cache. Lyapunov-LRU loads a nonresident expert only when its marginal gain exceeds the current loading price and uses global LRU eviction to maintain the fixed cache capacity.

Then we can define

$$r_{t,\ell}(n) = \frac{\sum_{e \in S_{t,\ell}(n)} q_{t,\ell}(e)}{D_{t,\ell}}, \qquad D_{t,\ell} = \sum_{e \in \mathrm{TopK}(q_{t,\ell})} q_{t,\ell}(e), \qquad g_{t,\ell}(n) = r_{t,\ell}(n) - r_{t,\ell}(0). \tag{2}$$

Here $D_{t,\ell}$ is the router mass of the native TopK route. The quantity $r_{t,\ell}(n)$ is the fraction of that mass retained after n loads, while $g_{t,\ell}(n)$ is the additional mass recovered relative to using the current cache without any new load. In our algorithm, we use $g_{t,\ell}(n)$ as the loading benefit available to the online controller. Note that it is a routing-based proxy, not a guarantee of answer quality.

3.2 Shared Virtual Queue and Loading Price

Following the virtual-queue approach in Lyapunov optimization (Neely, 2010), we maintain a shared virtual queue $Z \geq 0$ within each request to track loading pressure relative to the model-wide budget B. The queue is shared across all MoE layers and carried across decoding steps, allowing loading decisions at different layers and times to compete through a common resource signal.

Each decoding step contains L layer decisions, so the model-wide budget B corresponds to an allowance of B/L at each queue update. Starting from $Z_{0,0} = 0$, the queue evolves as

$$Z_{t,\ell+1} = \left[Z_{t,\ell} + a_{t,\ell} - \frac{B}{L} \right]_+ \text{ when } \ell < L-1, \text{ and } Z_{t+1,0} = \left[Z_{t,L-1} + a_{t,L-1} - \frac{B}{L} \right]_+ \tag{3}$$

The same queue is shared across all layers and carried across decoding steps. Hence, B/L is not a separate budget reserved for each layer: loading at one layer changes the queue state seen by subsequent layers.

We then convert this shared loading pressure into a loading price. For a control parameter $V > 0$, define

$$\lambda_{t,\ell} = \frac{L Z_{t,\ell}}{V}. \tag{4}$$

A larger $\lambda_{t,\ell}$ means that a missing expert must provide greater routing recovery to justify another load. Therefore, we let the controller choose

$$a_{t,\ell} = \arg\max_{n \geq 0} \left\{ g_{t,\ell}(n) - \lambda_{t,\ell} \cdot n \right\}, \tag{5}$$

breaking ties in favor of fewer loads.

We can see that the parameter V controls how strongly the accumulated loading pressure affects the current decision, and we use the same fixed V throughout our experiments. Section 4 analyzes the resulting queue dynamics and derives finite-request bounds on cumulative loading.

3.3 Cache Update and Execution

After choosing $a_{t,\ell}$, the controller loads the selected experts (i.e., the nonresident experts with top router probability) and updates the shared cache. Since we need to also remove $a_{t,\ell}$ experts, the controller locks the selected route $S_{t,\ell}(a_{t,\ell})$, and then evicts the $a_{t,\ell}$ globally least-recently-used experts, while keeping at least K cached experts in every layer. The released GPU slots are reused for the incoming weights, so the physical cache never exceeds its capacity LC. Because the cache is shared across layers, an expert may be evicted from another layer whenever that layer has more than K cached experts.

The controller may change which experts are executed, but it preserves the native route width K and the model's original mixture-weighting rule. The selected experts are therefore executed using the model-specific weighting rules summarized in Table 3. Cache recency is then updated for subsequent decisions. The full algorithm, Lyapunov-LRU, is described in Algorithm 1.

Algorithm 1 Lyapunov-LRU at one layer

Require: Current decoding step t, layer ℓ, router scores $q_{t,\ell}$, cache/recency state $R_{t,\ell}$, shared virtual queue $Z_{t,\ell}$

1: Compute candidate routes $S_{t,\ell}(n)$, loading benefit $g_{t,\ell}(n)$ and price $\lambda_{t,\ell} = LZ_{t,\ell}/V$
2: Choose the smallest maximizer of $g_{t,\ell}(n) - \lambda_{t,\ell} \cdot n$ and denote it by $a_{t,\ell}$
3: Add the $a_{t,\ell}$ selected incoming experts to a temporary resident set and lock the selected route $S_{t,\ell}(a_{t,\ell})$
4: Evict $a_{t,\ell}$ global LRU experts while preserving each layer's K-expert floor
5: Load the $a_{t,\ell}$ selected incoming experts into the released GPU slots and update cache recency
6: Update the shared queue for the next layer decision: $Z \leftarrow [Z_{t,\ell} + a_{t,\ell} - B/L]_+$

The model-specific implementation details are deferred to Section B.2.

4 Finite-Request Guarantee of Lyapunov-LRU

In this section, we prove an upper bound on the number of loads in Lyapunov-LRU, as stated following:

Proposition 1 (Bounded queue and finite-request loading)**.** *Under exact arithmetic, let $V > 0$, $B \geq 0$, and $Z_{0,0} = 0$. For any realized routing sequence, the decision and tie rule in Section 3 give, for a request with T controlled decoding steps,*

$$0 \leq Z_{t,\ell} \leq \frac{V}{L} + K, \qquad A \leq BT + \frac{V}{L} + K. \tag{6}$$

The queue bound holds at every layer decision and for the final queue value $Z_{T,0}$.

Since $g_{t,\ell}(n) \leq 1$, once $Z_{t,\ell} \geq V/L$ the price is at least one and the tie rule selects $n = 0$. Below this threshold, one layer decision increases the queue by at most K. Summing the queue updates then bounds cumulative loading. Thus, the excess over BT is bounded independently of T, and its per-step upper bound vanishes as T grows.

The full proof of Proposition 1 is deferred to Section A.

5 Experimental Setup

Models and datasets. We evaluate DeepSeek-V2-Lite-Chat, Qwen1.5-MoE-A2.7B-Chat, and Qwen3-30B-A3B (DeepSeek-AI et al., 2024; Qwen Team, 2024; Yang et al., 2025). Their MoE

layers contain $E = 64, 60, 128$ routed experts and select $K = 6, 4, 8$ experts per token, respectively. We use test subsets containing 200 questions each from GSM8K and original MBPP, and 160 questions each from MATH L1 and L5 (Cobbe et al., 2021; Austin et al., 2021; Hendrycks et al., 2021). All methods use the same selected questions. Sampling details are provided in Section B.4.

Generation and cache initialization. We use greedy decoding with BF16 weights, batch size one, each model's native chat template, and at most 8192 new tokens. All cached methods use the same prompt-based initialization: we record expert activations while processing the prompt, first select the K most frequently activated experts in each layer, and then fill the remaining global cache slots by activation frequency until LC experts are cached. Further generation and initialization details are given in Section B.

Baselines and comparison roles. Table 1 summarizes the reference methods.

Demand-LRU preserves the native route and loads every missing selected expert. It therefore measures the expert-loading cost of preserving native routing under a limited cache.

Uniform-LRU uses the same model-wide loading budget B as Lyapunov-LRU, but allocates loading opportunities uniformly across layers. Each layer accumulates B/L loading credits per decoding step and spends one credit on each missing native-route expert, in decreasing router-score order. This baseline tests whether dynamically allocating loading opportunities according to routing benefit is more effective than uniformly distributing the same configured target.

MoE-Infinity (Xue et al., 2024) uses expert-activation histories to guide caching and prefetching. Our Uniform-Infinity-derived baseline (Uniform-Inf.) keeps the Uniform-LRU admission rule but replaces LRU with history-based eviction. This baseline tests whether using a more complex eviction rule is helpful in caching.

Table 1: Baseline methods and their roles.

Method	Role
Demand-LRU	Native routing with LRU caching; every missing selected expert is loaded.
Uniform-LRU	Equal per-layer loading credits with global LRU eviction.
Uniform-Inf.	The same loading process as Uniform-LRU with history-based eviction inspired by MoE-Infinity (Xue et al., 2024)

Capacity and loading settings. For DeepSeek-V2-Lite-Chat and Qwen1.5-MoE-A2.7B-Chat, we use $C \in \{16, 32, 48\}$. For Qwen3-30B-A3B, we use $C \in \{32, 48\}$. Remaining parameters and complete settings are provided in Section B.

Metrics. We report answer accuracy, average expert loads per decoding step, and cumulative loads. MBPP accuracy is the fraction of questions for which the generated code passes all supplied tests. The average loading rate is the total number of expert loads across all layers divided by the total number of decoding steps. Physical evaluation also reports decode throughput (tokens/s) and CPU-to-GPU expert-weight transfers (MiB per decode token). Decode throughput is the total number of decode tokens divided by the corresponding total measured time. Detailed grading rules, metric definitions, and timing boundaries are provided in Sections B and D.

6 Generation Quality

In this section, we aim to address three questions. First, can a model preserve much of its generation quality while loading only a small fraction of the experts required by exact routing? Second, when loading is limited, does benefit-aware shared control use those loads effectively? Third, how does expert residency affect this trade-off?

6.1 Effect of Loading Budget at Fixed Cache Capacity

Figure 3 illustrates the budget–accuracy relationship using results at $C = 32$ for DeepSeek-V2-Lite-Chat and Qwen1.5-MoE-A2.7B-Chat on GSM8K, MBPP, and MATH L1. The displayed sweeps

vary $B \in \{2, 4, 8\}$ and compare Lyapunov-LRU with Uniform-LRU, and Uniform-Inf. Demand-LRU provides native-route accuracy references.

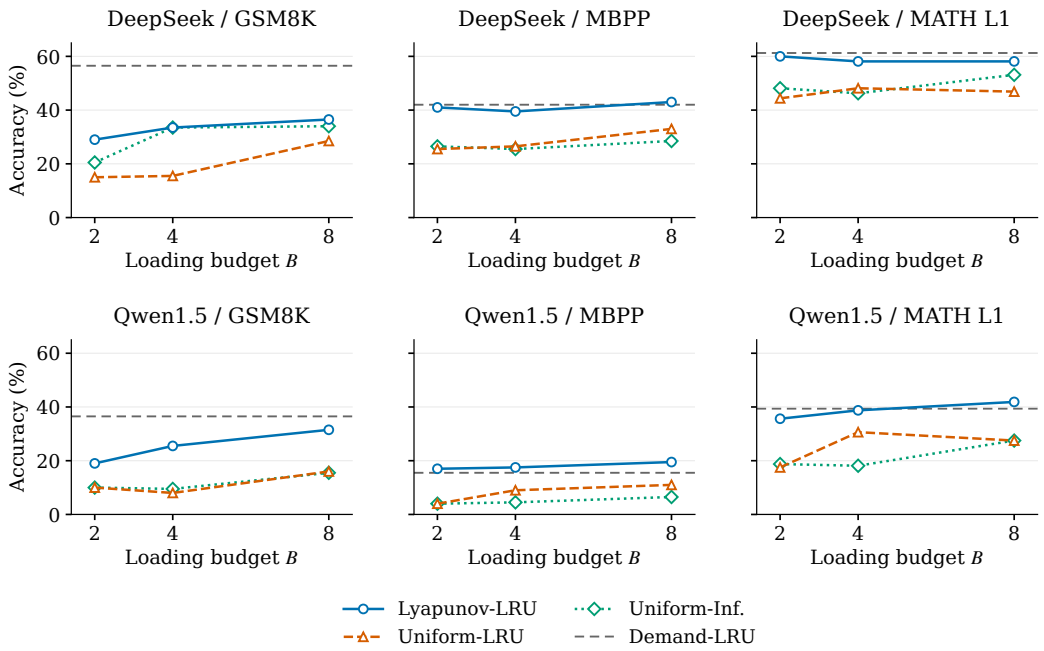

Figure 3: Selected budget sweeps for DeepSeek-V2-Lite-Chat and Qwen1.5-MoE-A2.7B-Chat on GSM8K, MBPP, and MATH L1. The horizontal axis is the loading budget B, with $C = 32$ fixed in these panels. Demand-LRU is a horizontal reference. GSM8K and MBPP use 200 questions per panel; MATH L1 uses 160.

The three tasks show different responses to the loading budget. On GSM8K, Lyapunov-LRU benefits from larger budgets over the displayed range and achieves higher accuracy than Uniform-LRU at the same configured budgets, while remaining below the native-route references. On MBPP, its accuracy varies less across budgets, with different trends for the two models. On MATH L1, Lyapunov-LRU remains close to the native-route references: DeepSeek-V2-Lite-Chat accuracy changes little across the displayed budgets, whereas Qwen1.5-MoE-A2.7B-Chat accuracy increases. Thus, a larger loading budget does not uniformly improve accuracy; its benefit depends on the model, task, and loading policy.

For GSM8K and MBPP at the displayed $C = 32, B = 4$ setting, Lyapunov-LRU reduces cumulative decode loads by 87.2%–92.4% relative to Demand-LRU, with accuracy differences of -23.0 to $+2.0$ percentage points. At the separate, larger-cache setting $C = 48, B = 4$, the corresponding load reductions are 64.0%–80.0%, with accuracy differences of -1.5 to $+3.0$ percentage points. The latter comparison is outside the displayed $C = 32$ sweeps; complete accuracy, loading-rate, and cumulative-count results are in Section C.

These results demonstrate that a model can preserve much of its generation quality with a small number of loadings: in most cases, Lyapunov-LRU has a comparable generation accuracy to Demand-LRU, while reducing the number of loadings significantly. Further, Lyapunov-LRU outperforms Uniform-LRU and Uniform-Inf. in most cases, indicating that benefit-aware shared control uses those loads effectively. Finally, in all these experiments, Uniform-Inf. achieves a similar performance as Uniform-LRU, suggesting that a more complex eviction rule is not very helpful.

6.2 Effect of Cache Capacity at a Fixed Loading Budget

Figure 4 illustrates the capacity–accuracy relationship with Qwen3-30B-A3B as an example from the broader evaluation. The displayed results compare $C \in \{32, 48\}$ on MATH L1, MATH L5, GSM8K, and MBPP, with $B = 2$ for Lyapunov-LRU and Uniform-LRU. Here C is average expert residency per layer. Demand-LRU still provides a native-route accuracy reference. Capacity comparisons for the other models and complete configuration results are provided in Section C.

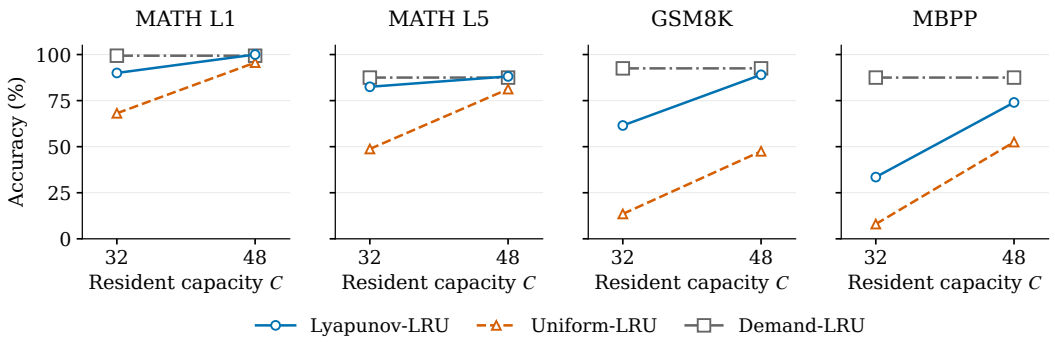

Figure 4: Selected capacity comparisons for Qwen3-30B-A3B on MATH L1, MATH L5, GSM8K, and MBPP. The horizontal axis is C; Lyapunov-LRU and Uniform-LRU use $B = 2$. Markers show the evaluated capacities, $C = 32, 48$, with connecting lines for readability. Demand-LRU has no loading budget and provides the native-route accuracy reference. MATH uses 160 questions per level; GSM8K and MBPP use 200 each.

Increasing cache capacity improves the accuracy of both budgeted policies on all four displayed tasks, with a particularly large improvement on MBPP. This is consistent with the intuition that a larger cache can provide more resident experts to be used without another load. At the larger capacity, Lyapunov-LRU approaches native-route accuracy more closely on most of the tasks, while a larger gap remains only on MBPP. The results illustrate how residency and loading interact: retaining more experts can make a small loading budget more useful.

At the displayed $C \in \{32, 48\}, B = 2$ setting, Lyapunov-LRU achieves higher accuracy than Uniform-LRU on all four tasks. This also demonstrates the effectiveness of dynamically allocating budgets during generation.

7 Physical Quality–Efficiency Trade-offs

Deployment setting. The BF16 weights of Qwen3-30B-A3B require approximately 61 decimal GB, exceeding the 48 GB memory of a single L20 GPU (Qwen Team, 2025; NVIDIA, 2026). We therefore use this deployment as a case study of running the model without keeping all weights in GPU memory. Expert weights are stored in CPU memory and copied to the GPU when needed. Each request uses one L20 GPU with caching size $C = 32$ and budget $B = 2$.

All methods are evaluated on the same 160 questions per MATH level. Table 2 uses the complete 160 questions per MATH level. Measurement details are given in Section D.

Accuracy and decode efficiency. At $C = 32, B = 2$, Lyapunov-LRU achieves 91.88% accuracy on MATH L1 and 83.13% on MATH L5. This result shows a slight decrease compared with Demand-LRU, which achieves 99.38% and 87.50% accuracy on MATH L1 and MATH L5, respectively. However, as a trade-off in accuracy, its decode throughput reaches 3.689 and 3.683 tokens/s, about 35% higher than the 2.720 and 2.662 tokens/s achieved by Demand-LRU. Moreover, the decode transfers per token of Lyapunov-LRU fall by 98.5% and 97.8%, respectively. For comparison, Uniform-LRU achieves similar decode throughput and decode transfers per token as Lyapunov-LRU, but its accuracy is much worse.

Table 2: Qwen3-30B-A3B offloading on L20 with 160 questions per MATH level. Cached methods use $C = 32$ and budgeted methods use $B = 2$. Accuracy, decode throughput, and decode-only transfers use the same question set in each row.

Level	Method	Accuracy (%)	Decode throughput (tokens/s)	Decode H2D (MiB/token)
L1	Demand-LRU	99.38	2.720	833.34
	Uniform-LRU	63.75	3.809	10.37
	Lyapunov-LRU	91.88	3.689	12.72
L5	Demand-LRU	87.50	2.662	750.57
	Uniform-LRU	60.00	3.677	13.09
	Lyapunov-LRU	83.13	3.683	16.55

8 Related Work

Offloading and memory management. MoE-Infinity uses expert-activation histories to guide caching and prefetching (Xue et al., 2024). Fiddler chooses between transferring expert weights for GPU execution and computing experts on the CPU (Kamahori et al., 2025), while WiSP jointly allocates GPU memory to expert weights and the KV cache (Zhang et al., 2026). These systems improve how the model's selected computation is served, but still preserve native routing, i.e., the system still requires executing every selected expert, even when its weights are not cached. In comparison, we consider an additional choice: replacing a missing expert with a resident alternative when its loading benefit is small. Given an expert-cache capacity, our controller decides which misses justify a transfer under a separate model-wide loading budget B. The distinction is therefore between serving native routing more efficiently and controlling loading through approximate routing.

Cache-aware and predictive routing. Cache-Prior reduces cache misses by adding a scaled bonus to resident experts' routing logits, while retaining the original mixture weights (Skliar et al., 2025). This provides a tunable cache preference, but does not provide an explicit control on the loading budget as Lyapunov-LRU. CommitMoE instead chooses to predict native routing ahead of the router (Li et al., 2026). This enables asynchronous expert loading to reduce time cost, but is very different from ours.

Online resource control. Lyapunov drift-plus-penalty provides a general framework for online decisions under average resource constraints (Neely, 2010). Stable-MoE applies this framework to distributed MoE training, jointly controlling token routing and computation resources under token-queue and energy constraints (Shi et al., 2026). They focus on the workload balance of experts, i.e., every expert should process neither too many nor too few tokens, which is a training-time balance rather than an inference-time one as in our work.

9 Conclusion

Memory-constrained MoE inference requires deciding both which experts to retain and which misses to load. In this paper, we propose a traning-free controller, Lyapunov-LRU, which combines marginal routing recovery with a shared online price, yielding tractable decisions and finite-request loading bounds. Across the evaluated models and tasks, we show that Lyapunov-LRU achieves strong accuracy while delivering significant speedups, demonstrating the efficiency of our method.

AI Use Statement

Generative AI tools, including ChatGPT and OpenAI Codex, were used to assist with manuscript editing, research discussion, implementation, and debugging. All technical content, code, experiments, and reported results were reviewed and verified by the authors.

Reproducibility Statement

Sections 3 and 4 specify the controller and its accounting assumptions. Sections B.4, C and D provide the implementation details, evaluation protocol, complete experimental results, and physical measurement methodology. Code and reproduction dependencies are included in the supplementary material.

Ethics Statement

This work studies inference on existing public mathematical and programming benchmarks. Reducing resource requirements can broaden access to models, but it can also make their existing misuse capabilities cheaper. Approximate routing can change model outputs, so its effect on generation quality depends on the task and operating point.

References

Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, et al. Program synthesis with large language models. *arXiv preprint arXiv:2108.07732*, 2021.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, et al. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021.

DeepSeek-AI et al. DeepSeek-V2: A strong, economical, and efficient mixture-of-experts language model. *arXiv preprint arXiv:2405.04434*, 2024. URL https://arxiv.org/abs/2405.04434.

William Fedus, Barret Zoph, and Noam Shazeer. Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity. *Journal of Machine Learning Research*, 23(120):1–39, 2022.

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.

Keisuke Kamahori, Tian Tang, Yile Gu, Kan Zhu, and Baris Kasikci. Fiddler: Cpu-gpu orchestration for fast inference of mixture-of-experts models. In *International Conference on Learning Representations*, volume 2025, pp. 56099–56115, 2025.

Han Li, Jingwei Sun, Junqing Lin, and Guangzhong Sun. CommitMoE: Efficient fallback-free MoE inference with offloading under GPU memory constraints. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 40, pp. 22904–22912, 2026. doi: 10.1609/aaai.v40i27.39454. URL https://ojs.aaai.org/index.php/AAAI/article/view/39454.

Michael Neely. *Stochastic network optimization with application to communication and queueing systems*. Morgan & Claypool Publishers, 2010.

NVIDIA. Ada Lovelace Architecture vGPU Types: NVIDIA L20. NVIDIA AI Enterprise Documentation, 2026. URL https://docs.nvidia.com/ai-enterprise/release-8/latest/infra-software/vgpu/reference/ada-lovelace.html. L20-48C framebuffer: 48 GB. Updated September 2, 2026; accessed September 25, 2026.

Qwen Team. Qwen1.5-MoE-A2.7B-Chat: Official model configuration. Hugging Face model repository, 2024. URL https://huggingface.co/Qwen/Qwen1.5-MoE-A2.7B-Chat/blob/222e0bec2bbfbd70d9ba5d1d7a777a81aad10859/config.json. Configuration accessed September 19, 2026; experiment checkpoint revision is recorded in the artifact.

Qwen Team. Qwen3-30B-A3B: Official model card. Hugging Face model repository, 2025. URL https://huggingface.co/Qwen/Qwen3-30B-A3B. Accessed September 21, 2026; model size and active parameter count.

Long Shi, Bingyan Ou, Kang Wei, Weihao Zhu, Zhe Wang, and Zhiyong Chen. Stable-moe: Lyapunov-based token routing for distributed mixture-of-experts training over edge networks. *IEEE Trans. Veh. Technol.*, 75(8):18815–18819, 2026. doi: 10.1109/TVT.2026.3676993. URL https://doi.org/10.1109/TVT.2026.3676993.

Andrii Skliar, Ties van Rozendaal, Romain Lepert, Todor Boinovski, Mart van Baalen, Markus Nagel, Paul Whatmough, and Babak Ehteshami Bejnordi. Mixture of cache-conditional experts for efficient mobile device inference. *Transactions on Machine Learning Research*, 2025. URL https://arxiv.org/abs/2412.00099v2.

Leyang Xue, Yao Fu, Zhan Lu, Luo Mai, and Mahesh Marina. Moe-infinity: Efficient moe inference on personal machines with sparsity-aware expert cache. *arXiv preprint arXiv:2401.14361*, 2024.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025.

Jiamu Zhang, Liang Wu, Mayank Darbari, and Liangjie Hong. Wisp: A working-set view of mixture-of-experts serving on extremely low-resource hardware. *arXiv preprint arXiv:2606.21868*, 2026.

A Decision Properties and Finite-Request Guarantees

This appendix establishes the local loading properties, proves the finite-request loading bound, and shows that the cache update is feasible.

A.1 Local Loading Decision

Fix the router probabilities and cache state at one layer decision. We omit the step and layer indices from q, D, r, g, S, and λ, and write R_ℓ for the current layer's resident set. Assume $|R_\ell| \geq K$, and let

$$m = \min\{K, E - |R_\ell|\}.$$

Lemma 1 (Optimal loading prefix and diminishing returns)**.** *For each integer $n \in \{0, \dots, m\}$, loading the n highest-probability nonresident experts maximizes the retained router mass among all sets of n incoming experts. Let $u_1 \geq \cdots \geq u_m$ be the m largest nonresident probabilities and $h_1 \geq \cdots \geq h_K$ the K largest resident probabilities. Then*

$$g(i) - g(i-1) = r(i) - r(i-1) = \frac{[u_i - h_{K-i+1}]_+}{D}, \qquad i = 1, \dots, m. \tag{7}$$

These marginal gains are nonincreasing. The smallest maximizing admission count in Equation (5) *is*

$$a = |\{i \in \{1, \dots, m\} : g(i) - g(i-1) > \lambda\}|. \tag{8}$$

Every incoming expert at this count belongs to the selected route $S(a)$. It is sufficient to consider the $m + 1 \leq K + 1$ loading counts $n = 0, \dots, m$.

Proof. At a fixed loading count, replacing an incoming expert with a higher-probability nonresident expert not yet selected cannot decrease the sum of the K largest probabilities in the candidate set. Repeating this exchange gives the sorted prefix.

If $u_i > h_{K-i+1}$, the previous $i - 1$ incoming experts also have probabilities greater than h_{K-i+1}. Before adding u_i, the selected route therefore contains these $i - 1$ experts and the $K - i + 1$ largest resident probabilities. Adding u_i replaces h_{K-i+1} and increases the retained mass by $u_i - h_{K-i+1}$. Otherwise, the $K - i + 1$ largest resident probabilities and the previous $i - 1$ incoming probabilities already provide K values at least as large as u_i, so the increase is zero. Dividing by D gives Equation (7). The gains are nonincreasing because u_i decreases while h_{K-i+1} increases.

The successive increments of $g(n) - \lambda n$ are $g(i) - g(i-1) - \lambda$. Since they are nonincreasing, accepting exactly the strictly positive increments gives the smallest maximizer. If $a > 0$, then $u_a > h_{K-a+1}$, so all a incoming experts belong to $S(a)$. Nonresident experts ranked below K cannot further increase the retained TopK mass. With a nonnegative price and the fewer-loads tie rule, loading more than K experts cannot improve the decision. Thus the counts $0, \dots, m$ suffice. □

Corollary 1 (Native-route recovery)**.** *At a fixed input with pairwise distinct router probabilities, each positive-gain prefix addition replaces a resident expert outside the native route with a missing native-route expert. At zero loading price, the selected route is $S^{\mathrm{nat}} = \mathrm{TopK}(q)$.*

Proof. Whenever $u_i > h_{K-i+1}$, at most $i - 1$ nonresident experts and $K - i$ resident experts have probabilities greater than u_i, so u_i belongs to the native route. The $K - i$ larger resident probabilities and the first i incoming probabilities all exceed h_{K-i+1}, so that resident expert lies outside the native route. The first m nonresident candidates contain every missing native-route expert. Taking all positive increments therefore attains the native TopK mass, since the remaining increments are zero. Distinct probabilities make the native route unique. □

A.2 Finite-Request Loading Bounds

Consider a request with T controlled decoding steps, indexed by $t = 0, \dots, T - 1$, and L layers, indexed by $\ell = 0, \dots, L - 1$. Prompt processing and cache initialization are excluded from the controlled decoding steps and admission counts. We assume exact arithmetic, $V > 0$, $B \geq 0$, and $D_{t,\ell} > 0$ at every layer decision.

Let $Z_{t,\ell}$ be the shared queue value before the loading decision at step t and layer ℓ. The queue starts from $Z_{0,0} = 0$ and follows Equation (3). We write $Z_{T,0}$ for the final queue value after the request. The per-step and total admission counts are

$$A_t = \sum_{\ell=0}^{L-1} a_{t,\ell}, \qquad A = \sum_{t=0}^{T-1} A_t.$$

Proposition 2 (Queue accounting)**.** *The cumulative admission count satisfies*

$$A \le BT + Z_{T,0}. \tag{9}$$

Proof. Since $[x]_+ \ge x$, the queue updates in Equation (3) imply

$$a_{t,\ell} \le \frac{B}{L} + Z_{t,\ell+1} - Z_{t,\ell}, \qquad 0 \le \ell < L-1,$$

and, at the last layer,

$$a_{t,L-1} \le \frac{B}{L} + Z_{t+1,0} - Z_{t,L-1}.$$

Summing these L inequalities within decoding step t cancels the intermediate queue values and gives

$$A_t \le B + Z_{t+1,0} - Z_{t,0}.$$

Summing over all T controlled decoding steps and using $Z_{0,0} = 0$ yields

$$\begin{aligned} A &\le \sum_{t=0}^{T-1} (B + Z_{t+1,0} - Z_{t,0}) \\ &= BT + Z_{T,0} - Z_{0,0} \\ &= BT + Z_{T,0}. \end{aligned}$$

□

Proof of Proposition 1. Queue values are nonnegative by Equation (3). By Equation (2), the loading benefit satisfies

$$g_{t,\ell}(0) = 0, \qquad 0 \le g_{t,\ell}(n) \le 1$$

for every feasible loading count n.

If $Z_{t,\ell} \ge V/L$, the loading price satisfies $\lambda_{t,\ell} = LZ_{t,\ell}/V \ge 1$. For every feasible $n \ge 1$,

$$g_{t,\ell}(n) - \lambda_{t,\ell} n \le 1 - \lambda_{t,\ell} n \le 0 = g_{t,\ell}(0).$$

The tie rule in Equation (5) therefore selects $a_{t,\ell} = 0$. The next queue value is then

$$\left[Z_{t,\ell} - \frac{B}{L} \right]_+ \le Z_{t,\ell},$$

so the queue cannot increase.

If $Z_{t,\ell} < V/L$, Lemma 1 gives $a_{t,\ell} \le K$. The next queue value satisfies

$$\left[Z_{t,\ell} + a_{t,\ell} - \frac{B}{L} \right]_+ \le Z_{t,\ell} + a_{t,\ell} \le Z_{t,\ell} + K < \frac{V}{L} + K.$$

These two cases apply both to updates within a decoding step and to the update from its last layer to the next step.

Starting from $Z_{0,0} = 0$, induction over the layer decisions in execution order gives

$$0 \le Z_{t,\ell} \le \frac{V}{L} + K$$

at every layer decision. The same bound holds for the final queue value $Z_{T,0}$. Combining it with Equation (9) yields

$$A \le BT + Z_{T,0} \le BT + \frac{V}{L} + K.$$

The argument holds for every realized sequence of router probabilities and cache states. □

The bound also applies to $B = 0$, allowing a finite number of admissions rather than requiring zero admissions at every step.

A.3 Feasibility of Floor-Safe Eviction

Lemma 2 (Feasibility of floor-safe eviction)**.** *Suppose the cache contains LC experts and at least K experts in every layer. Let a be the selected admission count at the current layer. After adding the a incoming experts to a temporary logical set, it is possible to evict a old cached experts while protecting $S(a)$ and preserving each layer's K-expert floor. The resulting cache has size LC and contains $S(a)$.*

Proof. By Lemma 1, every incoming expert belongs to $S(a)$ and is protected. Thus any unprotected expert belongs to the old cache. Before all a removals are complete, the temporary set contains more than $LC \geq LK$ experts, so some layer contains more than K experts. If this is the current layer, at least one expert lies outside its protected K-expert route. Otherwise, all experts in that layer are unprotected. In either case, an eligible victim exists whose removal preserves the layer's floor. Repeating this argument completes all a removals, including when the least-recently-used eligible expert is chosen at each removal.

The final set has size LC, contains $S(a)$, and retains at least K experts in every layer. Since all victims belong to the old cache, the incoming weights can be loaded into the released GPU slots without exceeding the cache capacity LC. □

B Implementation and Evaluation Protocol

B.1 Control Parameters and Initialization

All reported Lyapunov-LRU experiments use $V = 3500$, corresponding to an effective utility scale of V/L at each layer. Uniform-LRU starts with zero loading credits in every layer. At the start of each controlled decoding forward, each layer receives B/L credits, with its accumulated balance capped at K. Unused credits carry forward, and each missing native-TopK expert consumes one credit. Experts are loaded in decreasing router-probability order. All cached policies use the same global capacity LC, a minimum of K resident experts per layer, and protection of the current execution set during eviction.

Each request is initialized independently from prompt routing. We first reserve the K most frequently activated experts in each layer, then fill the remaining global cache slots by activation frequency. Ties are broken by accumulated routing mass and then deterministic expert order. Last-access times during prompt processing initialize LRU recency. The cache, virtual queue, loading credits, and activation history are reset between requests. Prompt processing and cache initialization are excluded from the controlled decoding count T and admission count A, and are outside the decoding budget B. Their costs are included in the generation time and complete-request transfers of the physical experiments.

B.2 Native Mixture Weights

For a selected execution set S, we use the model-specific mixture-weighting rules in Table 3, where q_e is the router probability of expert e before restricting the route to S. DeepSeek-V2-Lite-Chat and Qwen1.5-MoE-A2.7B-Chat do not renormalize probabilities within the selected set; DeepSeek-V2-Lite-Chat additionally applies its native scaling factor α. Qwen3-30B-A3B renormalizes the selected probabilities to sum to one. Shared expert branches remain unchanged and are excluded from cache capacity and admission counts.

Table 3: Model-specific execution weights for a selected route S. The probabilities q are computed over all routed experts.

Model	(L, E, K)	Routed weight, $e \in S$	Shared branch
DeepSeek-V2-Lite-Chat	$(26, 64, 6)$	αq_e	Native shared experts
Qwen1.5-MoE-A2.7B-Chat	$(24, 60, 4)$	q_e	Sigmoid-gated shared expert
Qwen3-30B-A3B	$(48, 128, 8)$	$q_e / \sum_{j \in S} q_j$	None

At a fixed hidden state, preserving the native TopK set and its mixture weights preserves the routed contribution in exact arithmetic. Budgeted expert substitution can change this contribution by changing the selected experts.

B.3 History-Based Eviction Baseline

Uniform-Inf. uses the same loading credits, global cache capacity, execution-set protection, and per-layer residency floors as Uniform-LRU. Its history-based eviction rule is inspired by MoE-Infinity (Xue et al., 2024).

The baseline collects native-TopK activation counts in 32-token windows and retains at most 64 historical windows within each request, including the final partial prompt window. Counts are normalized within each layer. The current observed activation pattern is matched to the history using cosine similarity, with ties resolved in favor of the oldest window. When a usable match exists, its normalized activation frequencies are used for eviction; otherwise, the current pattern is used. Unobserved layers use uniform probabilities.

Let $p_{\ell,e}$ denote the normalized activation frequency assigned to expert e in layer ℓ by the pattern used for eviction. Its retention score is

$$p_{\ell,e}\left(1-\frac{\ell}{L}\right), \qquad \ell = 0, \ldots, L-1.$$

Among experts eligible for eviction, the lowest-scoring expert is removed first, with ties broken by LRU recency and then expert order. Uniform-Inf. is a request-local adaptation: it uses neither prefetching nor cross-request history and is evaluated only in the logical experiments.

B.4 Cohorts and Generation

GSM8K uses a 200-question subset of the 1,319-record main test split. MBPP uses a 200-question subset of the original test IDs 11–510, sorted by source ID before sampling. For each dataset, we apply Python's local `random.Random(0).shuffle` and select the first 200 records.

MATH uses the ModelScope MATH test archive. We preserve the fixed test-file order, filter by difficulty level, and take the first 160 records from each of L1 and L5 without shuffling or subject stratification.

All methods and models use the same selected question identities and order, while retaining their own tokenizers and chat templates.

Generation and stopping. We use greedy decoding with BF16 weights, batch size one, native chat templates, a 2,048-token input limit without truncation, and at most 8,192 new tokens. There is no answer-detection stopping rule. Qwen3-30B-A3B retains thinking mode.

DeepSeek-V2-Lite-Chat and Qwen3-30B-A3B process the prompt prefix followed by the final prompt token using native routing. Qwen1.5-MoE-A2.7B-Chat uses one whole-prompt native pass for every method. In both schedules, the final prompt position produces the first output token. Cache residency restrictions begin when that output is used to predict the next token. A generation of G tokens normally involves $G-1$ controlled decoding forwards. Loading rates use the recorded `processed_router_tokens` count T. An emitted EOS token counts toward G, even when omitted from the decoded text.

The generation loop stops on the tokenizer's single EOS token. For Qwen3-30B-A3B, this is token ID 151645, whereas the checkpoint's default generation configuration also accepts 151643. All compared methods use the same stated stopping rule for a given model; Qwen3-30B-A3B runs do not use the checkpoint's full default EOS set.

B.5 Scoring and Aggregation

GSM8K requires a single final numeric answer line marked `####` and is graded by exact rational comparison. MATH uses the fixed answer extractor and symbolic-equivalence grader. Accuracy is the fraction of questions accepted by the corresponding scorer.

MBPP prompts contain the task and its three supplied tests, without demonstration solutions. Extracted code is executed against these tests in a version-pinned, resource-limited Docker image, with a 10-second timeout and no GPU. MBPP accuracy is the fraction of questions whose generated code passes all supplied tests under this zero-shot protocol. Parse failures, incorrect outputs, and scored execution timeouts remain in the denominator.

Qwen3-30B-A3B MBPP final-code extraction. For Qwen3-30B-A3B, to separate the final code from preceding thinking text, we extract the suffix after the last generated thinking-close token (ID 151668), or the full generated sequence if this token is absent. The same code extractor and supplied tests are then applied to the resulting text. This extraction rule is applied uniformly to all reported Qwen3-30B-A3B MBPP configurations, each containing 200 questions. Scoring for DeepSeek-V2-Lite-Chat, Qwen1.5-MoE-A2.7B-Chat, and GSM8K remains unchanged.

Aggregation. Accuracy, loading rate, and cumulative admissions are computed from complete matched question sets. For question i, let A_i and T_i be its cumulative decode admissions and controlled decoding length. The pooled loading rate is

$$\rho = \frac{\sum_i A_i}{\sum_i T_i},$$

not the unweighted mean of per-question loading rates. Reaching the generation cap is recorded as an additional flag rather than a separate failure category. Capped and unparsed generations remain in the reported accuracy denominator.

C CORE EXPERIMENTAL RESULTS

The tables below report 216 logical configurations: 24 per task for each of DeepSeek-V2-Lite-Chat and Qwen1.5-MoE-A2.7B-Chat, and six per task for Qwen3-30B-A3B. Each configuration uses 160 questions for MATH L1 or L5 and 200 questions for GSM8K or MBPP, yielding 38,880 sample–configuration outcomes. Logical experiments run on A40 for DeepSeek-V2-Lite-Chat and Qwen1.5-MoE-A2.7B-Chat, and on H200 for Qwen3-30B-A3B. All model weights remain in GPU memory, while cache residency and expert admissions are tracked logically. Each method executes its selected experts and generates its own answers, allowing evaluation of the quality–loading trade-off. Physical offloading results are reported separately in Section 7.

Correct denotes the number of accepted answers under the scoring rules in Section B; dividing it by the number of questions gives accuracy. For each configuration, $A = \sum_i A_i$ is the cumulative number of decode admissions across all controlled MoE layers, and $T = \sum_i T_i$ is the total number of controlled decoding steps, where i indexes questions. The pooled loading rate is $\rho = A/T = \sum_i A_i / \sum_i T_i$, rather than the mean of per-question rates. Prompt processing and cache initialization are excluded from these decode counters. Uniform-Inf. denotes the history-based eviction baseline described in Section B.3.

Table 4: Complete logical results for DeepSeek-V2-Lite-Chat and Qwen1.5-MoE-A2.7B-Chat on MATH L1. Each model has 160 questions per configuration; all 48 configurations are included. $\rho = A/T$ is average loads per decode forward; A is cumulative loads.

			DeepSeek-V2-Lite-Chat			Qwen1.5-MoE-A2.7B-Chat		
Method	C	B	Correct	ρ	A	Correct	ρ	A
Demand-LRU	16	–	98	89.837	6,079,596	63	53.445	2,801,143
Lyapunov-LRU	16	4	71	3.717	816,390	37	4.089	212,737
Uniform-LRU	16	4	22	2.922	1,643,552	13	3.608	707,223
Uniform-Inf.	16	4	35	2.690	1,164,466	15	3.531	538,742
Demand-LRU	32	–	98	50.808	3,438,363	63	30.281	1,587,110
Lyapunov-LRU	32	2	96	2.067	157,193	57	2.018	111,807
Lyapunov-LRU	32	4	93	4.070	227,181	62	3.683	227,664
Lyapunov-LRU	32	8	93	7.102	525,884	67	8.028	419,589
Uniform-LRU	32	2	71	1.597	291,320	28	1.498	168,158
Uniform-LRU	32	4	77	3.444	433,944	49	3.195	247,061
Uniform-LRU	32	8	75	7.306	708,579	44	7.960	447,016
Uniform-Inf.	32	2	77	1.636	253,377	30	1.919	147,463
Uniform-Inf.	32	4	74	3.273	387,152	29	3.773	268,242
Uniform-Inf.	32	8	85	6.874	804,370	44	7.869	379,637
Demand-LRU	48	–	98	16.562	1,120,795	63	10.525	551,616
Lyapunov-LRU	48	2	101	1.702	112,980	74	2.057	97,273
Lyapunov-LRU	48	4	108	3.555	245,372	67	3.373	190,209
Lyapunov-LRU	48	8	97	6.948	328,247	87	6.869	303,137
Uniform-LRU	48	2	87	1.953	127,469	51	1.888	94,131
Uniform-LRU	48	4	94	3.849	224,843	67	3.853	235,916
Uniform-LRU	48	8	93	7.701	335,160	67	7.224	358,057
Uniform-Inf.	48	2	97	1.954	100,074	63	1.942	114,413
Uniform-Inf.	48	4	91	3.769	265,286	68	3.640	197,572
Uniform-Inf.	48	8	91	7.681	377,043	72	4.704	407,586

Table 5: Complete logical results for DeepSeek-V2-Lite-Chat and Qwen1.5-MoE-A2.7B-Chat on MATH L5. Each model has 160 questions per configuration; all 48 configurations are included. $\rho = A/T$ is average loads per decode forward; A is cumulative loads.

			DeepSeek-V2-Lite-Chat			Qwen1.5-MoE-A2.7B-Chat		
Method	C	B	Correct	ρ	A	Correct	ρ	A
Demand-LRU	16	–	5	89.951	15,096,807	8	55.325	7,502,299
Lyapunov-LRU	16	4	2	3.937	2,304,105	5	3.887	862,974
Uniform-LRU	16	4	1	3.167	2,802,946	1	3.467	1,429,693
Uniform-Inf.	16	4	4	2.767	2,285,168	3	3.545	1,748,797
Demand-LRU	32	–	5	48.846	8,197,958	8	27.266	3,697,436
Lyapunov-LRU	32	2	2	1.942	394,666	4	1.839	299,045
Lyapunov-LRU	32	4	8	3.767	578,058	5	3.680	557,953
Lyapunov-LRU	32	8	2	7.962	1,154,692	7	7.286	1,221,718
Uniform-LRU	32	2	7	1.518	827,644	6	1.670	434,156
Uniform-LRU	32	4	4	3.470	1,478,779	5	3.254	820,231
Uniform-LRU	32	8	4	7.376	2,879,433	5	6.448	980,455
Uniform-Inf.	32	2	5	1.767	865,539	1	1.823	430,701
Uniform-Inf.	32	4	5	3.530	1,248,436	5	3.707	765,911
Uniform-Inf.	32	8	6	7.322	1,722,806	6	6.686	1,080,576
Demand-LRU	48	–	5	16.769	2,814,404	8	9.142	1,239,664
Lyapunov-LRU	48	2	5	1.897	374,748	4	1.664	265,900
Lyapunov-LRU	48	4	5	3.557	772,432	5	3.384	419,817
Lyapunov-LRU	48	8	6	6.242	1,173,088	7	4.756	804,626
Uniform-LRU	48	2	5	1.892	331,324	6	1.515	220,032
Uniform-LRU	48	4	8	3.664	872,380	3	3.313	393,520
Uniform-LRU	48	8	5	6.967	1,313,527	10	5.559	755,856
Uniform-Inf.	48	2	1	1.786	251,351	6	1.628	230,240
Uniform-Inf.	48	4	6	3.506	749,639	1	2.909	551,350
Uniform-Inf.	48	8	6	6.442	1,051,447	7	5.634	869,256

Table 6: Complete logical results for DeepSeek-V2-Lite-Chat and Qwen1.5-MoE-A2.7B-Chat on GSM8K. Each model has 200 questions per configuration; all 48 configurations are included. $\rho = A/T$ is average loads per decode forward; A is cumulative loads.

			DeepSeek-V2-Lite-Chat			Qwen1.5-MoE-A2.7B-Chat		
Method	C	B	Correct	ρ	A	Correct	ρ	A
Demand-LRU	16	–	113	88.530	6,079,423	73	64.675	3,121,858
Lyapunov-LRU	16	4	20	3.887	997,512	16	4.208	177,858
Uniform-LRU	16	4	3	2.689	2,239,282	5	3.269	571,133
Uniform-Inf.	16	4	13	2.992	1,369,437	6	3.428	380,594
Demand-LRU	32	–	113	50.937	3,497,891	73	37.071	1,789,417
Lyapunov-LRU	32	2	58	2.074	163,698	38	2.183	104,470
Lyapunov-LRU	32	4	67	4.055	307,480	51	4.126	210,340
Lyapunov-LRU	32	8	73	8.037	499,919	63	8.078	414,607
Uniform-LRU	32	2	30	1.938	181,435	20	1.723	114,099
Uniform-LRU	32	4	31	2.824	240,285	16	3.957	197,529
Uniform-LRU	32	8	57	7.667	803,333	32	7.965	438,175
Uniform-Inf.	32	2	41	1.732	155,679	20	1.664	120,836
Uniform-Inf.	32	4	67	3.919	259,782	19	3.850	227,709
Uniform-Inf.	32	8	68	7.925	467,194	31	7.944	372,803
Demand-LRU	48	–	113	18.818	1,292,252	73	13.221	638,196
Lyapunov-LRU	48	2	97	2.050	132,021	67	1.827	119,060
Lyapunov-LRU	48	4	110	4.010	258,230	73	3.465	194,648
Lyapunov-LRU	48	8	111	7.457	495,635	80	7.174	364,524
Uniform-LRU	48	2	87	1.821	149,604	45	1.688	99,690
Uniform-LRU	48	4	81	3.906	344,394	39	3.491	248,573
Uniform-LRU	48	8	96	7.854	480,929	69	7.249	427,523
Uniform-Inf.	48	2	74	1.957	138,694	51	1.699	98,719
Uniform-Inf.	48	4	79	3.936	282,842	57	3.388	201,113
Uniform-Inf.	48	8	100	6.244	497,097	55	7.544	385,351

Table 7: Complete logical results for DeepSeek-V2-Lite-Chat and Qwen1.5-MoE-A2.7B-Chat on MBPP. Each model has 200 questions per configuration; all 48 configurations are included. $\rho = A/T$ is average loads per decode forward; A is cumulative loads.

			DeepSeek-V2-Lite-Chat			Qwen1.5-MoE-A2.7B-Chat		
Method	C	B	Correct	ρ	A	Correct	ρ	A
Demand-LRU	16	–	84	94.954	1,294,126	31	50.568	2,585,134
Lyapunov-LRU	16	4	60	3.484	101,987	28	4.253	132,348
Uniform-LRU	16	4	14	2.669	475,192	3	2.023	230,337
Uniform-Inf.	16	4	13	2.070	338,988	2	2.322	269,267
Demand-LRU	32	–	84	54.182	738,449	31	27.320	1,396,643
Lyapunov-LRU	32	2	82	2.421	34,326	34	2.189	91,036
Lyapunov-LRU	32	4	79	4.325	56,300	35	4.137	178,705
Lyapunov-LRU	32	8	86	8.204	119,503	39	8.020	345,323
Uniform-LRU	32	2	51	1.371	39,250	8	1.672	95,650
Uniform-LRU	32	4	53	2.211	119,493	18	3.671	163,152
Uniform-LRU	32	8	66	7.886	160,518	22	6.425	268,546
Uniform-Inf.	32	2	53	1.101	40,956	8	1.465	65,628
Uniform-Inf.	32	4	51	1.777	52,541	9	2.972	146,925
Uniform-Inf.	32	8	57	7.794	106,256	13	7.911	287,297
Demand-LRU	48	–	84	19.635	267,604	31	8.963	458,194
Lyapunov-LRU	48	2	92	2.199	31,409	39	2.083	89,505
Lyapunov-LRU	48	4	90	3.998	57,698	37	3.873	164,777
Lyapunov-LRU	48	8	86	7.700	103,825	42	4.744	274,620
Uniform-LRU	48	2	85	1.774	23,333	29	1.928	78,734
Uniform-LRU	48	4	79	3.740	48,382	27	3.833	164,319
Uniform-LRU	48	8	88	7.532	106,252	34	6.946	288,246
Uniform-Inf.	48	2	82	1.863	39,692	21	1.926	82,080
Uniform-Inf.	48	4	77	3.732	48,645	24	3.846	161,050
Uniform-Inf.	48	8	82	7.544	100,326	34	6.954	284,911

Table 8: Complete Qwen3-30B-A3B MATH logical core. Each task has 160 questions per configuration; all 12 configurations are included. $\rho = A/T$ is average loads per decode forward; A is cumulative loads.

			MATH L1			MATH L5		
Method	C	B	Correct	ρ	A	Correct	ρ	A
Demand-LRU	32	–	159	92.559	24,497,936	140	84.154	62,577,896
Lyapunov-LRU	32	2	144	1.431	491,683	132	1.846	1,550,372
Uniform-LRU	32	2	109	1.197	836,500	78	1.348	1,409,681
Demand-LRU	48	–	159	44.095	11,670,720	140	36.022	26,786,718
Lyapunov-LRU	48	2	160	2.005	539,059	141	1.998	1,523,125
Uniform-LRU	48	2	153	1.622	503,463	130	1.768	1,564,197

Table 9: Complete Qwen3-30B-A3B GSM8K/MBPP logical core. Each task has 200 questions per configuration; all 12 configurations are included. $\rho = A/T$ is average loads per decode forward; A is cumulative loads.

			GSM8K			MBPP		
Method	C	B	Correct	ρ	A	Correct	ρ	A
Demand-LRU	32	–	185	98.309	17,058,638	175	88.718	43,798,262
Lyapunov-LRU	32	2	123	1.623	588,157	67	1.244	1,232,581
Uniform-LRU	32	2	27	1.564	1,032,634	16	0.735	1,045,271
Demand-LRU	48	–	185	43.317	7,516,363	175	41.604	20,538,996
Lyapunov-LRU	48	2	178	1.858	333,277	148	1.913	1,151,309
Uniform-LRU	48	2	95	1.514	502,504	105	1.532	1,169,260

D PHYSICAL MEASUREMENT DETAILS

D.1 TIMING AND TRANSFER ACCOUNTING

For request i, W_i is the custom generation routine's wall time. It includes prompt processing, cache initialization, model forwards, expert loading, controller work, synchronization, and loop bookkeeping. It excludes checkpoint loading, tokenization, outer request preparation, final text decoding, and scoring. Let τ_i be the sum of the recorded prefill, first-token bootstrap, and initialization times. The aggregate decode throughput is

$$Q_{\mathrm{dec}} = \frac{\sum_i (G_i - 1)}{\sum_i (W_i - \tau_i)} \quad \text{tokens/s.}$$

The residual time includes decode computation, control, staging, copies, and synchronization, together with remaining loop overhead. Controller and copy intervals can occur inside model-forward timers, so those timers are not added separately. In all L20 records, the controlled-forward count is $T_i = G_i - 1$.

The logical experiments in Section C and the physical experiments here are independent end-to-end generations on different hardware and execution paths, so their greedy decoding trajectories need not be token-identical. Accordingly, all physical comparisons are made under the same L20 offloading implementation and measurement setup.

D.2 A40 COPY MEASUREMENT FOR FIGURE 1

The transfer-only example uses synthetic 9-MiB expert-sized buffers on one A40. Packed batches of $U \in \{1, 2, 4, 8, 16, 32, 64\}$ buffers are copied from pinned host memory to preallocated GPU storage on one asynchronous CUDA stream. Each size has 20 warm-ups and 100 timed repetitions. Synchronized CUDA events exclude allocation and buffer filling. The figure shows medians and within-size 10th–90th percentiles. An equal-weight affine fit to the seven medians gives

$$\widehat{t}_{\mathrm{copy}}(U) = 0.02053 + 0.35245U \text{ ms.} \tag{10}$$

This microbenchmark illustrates the dependence of copy time on the number of expert loads.