MARCAP: FROM MARGINAL ROUTING PROBABILITIES TO RISK-BOUNDED MOE WORKSPACE PROVISIONING WITH LOSSLESS REPLAY

Anonymous authors
Paper under double-blind review

ABSTRACT

MoE kernels reserve workspace before routing loads are known. Reserving for the worst case wastes memory, while fixed capacity factors fail to quantify overflow risk. We present MarCap (**mar**ginal-based **cap**acity provisioning), which maps routing marginals and a user-specified overflow risk budget to per-rank receive-buffer capacities. It optimizes over all possible fixed top-k joint routing laws consistent with the marginals, covering arbitrary within-token expert co-selection correlations. Under the IID token assumption, we formulate receive-buffer capacity bounds in terms of extremal moment-generating functions and use exact feasibility conditions to reduce their computation to a small linear program that is solved efficiently by a lower convex-hull construction. We validate MarCap on 2.737B token-level routing records from two LLMs. Analysis identifies inter-token dependence in prefill and marginal drift as sources of potential undersizing. Traffic mixing reduces dependence-related shortfalls, while sliding-window recalibration mitigates drift-related undersizing. These results clarify the theory's practical scope. We implement MarCap in `megamoe_replay`, our fused MoE operator with bounded workspace and lossless replay. Across 660k EP4 invocations on Ascend 950PR, `megamoe_replay` preserves reference outputs and reduces scratch workspace requirements by 44.1–71.4% versus full reservation across the evaluated configurations. In the decode workloads tested on hardware, which are well aligned with the IID model, MarCap keeps the observed frequency of overflow which would trigger additional replay rounds within the user-specified risk budget. Experimental data confirm the resulting benefit: substantial scratch workspace savings with mean latency comparable to the worst-case reservation, with a median absolute difference of 0.36% across the tested decode configurations.

1 INTRODUCTION

Expert parallelization in the deployment of Mixture-of-Experts (MoE) models faces several dilemmas that originate from the stochastic nature of the token-dispatching process. Besides causing compute under-utilization, potential load imbalances also complicate receive-buffer provisioning. This is because this temporary workspace, as part of *scratch memory* in systems terminology, must be preallocated in HBM before the kernel call rather than resized according to each invocation's realized routing load to avoid frequent host intervention and facilitate graph capture and replay with stable buffer addresses.

In the conventional token-dispatch process, each token selects k experts, some of which could reside on the same accelerator card (rank). In this paper, we mainly focus on the case where there is no token deduplication, so each token–expert assignment occupies exactly one row in the receive buffer. We measure per-rank capacity using these *assignment rows*, meaning a token that selects two experts on the same rank contributes two rows instead of one. The mathematical theory and experimental results for token deduplication and redundant experts are included in Appendices D and E.

Always using worst-case reservation guarantees sufficient space and avoids token dropping, but can waste substantial amounts of precious HBM. In comparison, settling on a heuristic-based fixed capacity factor can indeed reduce this reservation tax, but an undersized buffer can sacrifice output

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.

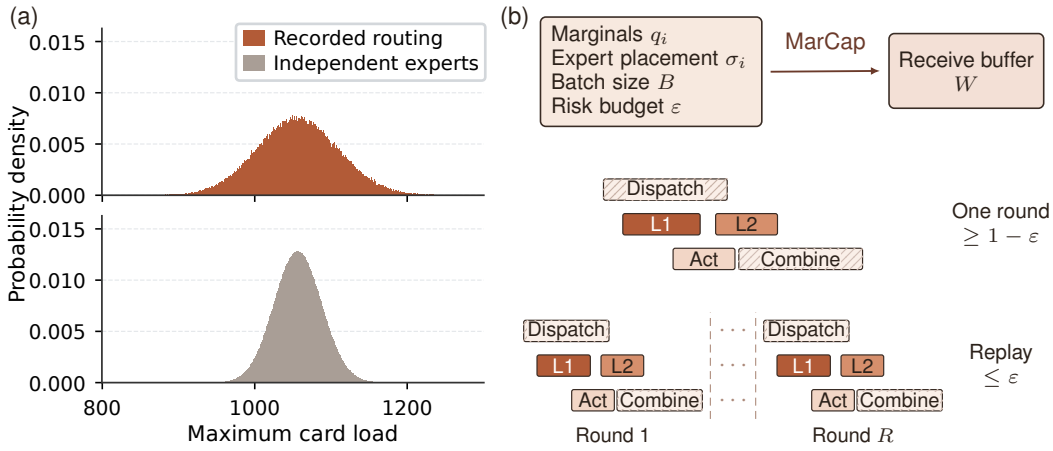


Figure 1: **Max card-load distribution and MarCap integration with `megamoe_replay`.** (a) Max card-load densities for Qwen Layer 19 with EP8 and global batch 512 with fixed expert placement: 200000 batches sampled from recorded decode token routes with replacement versus the distribution computed assuming inter-expert independence with matched marginals. Experimental details are given in Appendix F. (b) Mechanism of MarCap integrated with the multi-round lossless replay MoE kernel `megamoe_replay`.

quality when overflowing tokens are dropped (Lepikhin et al., 2021; Fedus et al., 2022), and the factor does not by itself quantify overflow risk. Thus, we seek to answer a basic question: *what receive-buffer capacity keeps the overflow probability within a user-specified budget?* Our goal is to reduce buffer reservation while avoiding token dropping through multi-round replay when needed, without incurring substantial replay overhead.

Practically, one could claim that direct empirical tail estimation could give precisely that capacity estimation. However, calibration becomes costly when risk budgets are intentionally chosen to be extremely small, because observing a constant expected number of overflow events with probability ε requires on the order of $\frac{1}{\varepsilon}$ independent batch observations, and once the experts are rearranged onto different ranks, new calibration observations are generally needed unless sufficient joint routing information is retained.

Expert selection frequencies provide another practical starting point. Existing routing telemetry records how often each expert is selected, and these marginals can be reused after changing the batch size or expert placement. Such statistics are already used in expert load balancing (EPLB), including expert rearrangement (DeepSeek-AI, 2026b). However, because marginals pool observations across individual tokens, there exists an important gap as they do not reveal which experts a token selects together, effectively masking the within-token expert selection correlations. The same marginals can produce very different tail behavior in per-card loads despite identical mean loads (Figure 1(a)). Therefore, a theory based only on these marginals must account for all compatible fixed top-k joint routing laws instead of assuming independent expert selections.

Our algorithm, MarCap, computes a receive-buffer capacity from the expert selection marginals, expert placement, batch size, and a user-specified overflow probability budget. It first computes the worst-case moment-generating function (MGF) over all top-k routing distributions consistent with the supplied marginals. Our central result (Theorem 1) gives necessary and sufficient conditions for a per-card count distribution to be realizable. These conditions reduce the original linear program over $\binom{E}{k}$ expert combinations to one with at most $k+1$ count probabilities as variables, without relaxing the original optimization. Under the IID token assumption, the resulting MGF bound gives a capacity bound for the aggregate batch load through the Chernoff inequality.

We then make the computation highly efficient through two further improvements. Even after the reduction above that makes the linear program small, the naive approach would still demand repeated LP solves during the search over the parameter θ to obtain the smallest sufficient buffer size. As a result, the actual implementation of the algorithm could still be computationally demanding. The next improvement comes from the realization that given fixed marginals and expert placement, there exists

a universal count distribution that maximizes the MGF for every positive θ (Theorem 2), eliminating the need for repeated LP solves while reducing the total number of linear programs to only one for each rank, and this improvement alone has produced an 11.3–$20.1\times$ speedup over the naive search version. Finally, we make a subsequent improvement after realizing that a lower-hull method can construct the same maximizing distribution directly without the need to solve any linear program (Section 2.3), and the same distribution can be reused across different batch sizes and risk budgets. In the end, the most efficient hull based computation has produced a 94.4–$109.9\times$ speedup over the naive version, greatly reducing the computational cost for the implementation of MarCap.

We use routing records from 2.7 billion tokens across the prefill and decode stages of two different LLMs to validate our theory under different scenarios. In selected shuffled-prefill and decode settings, MarCap achieves receive-buffer capacity savings of 84.2–95.5% relative to worst-case reservation across risk budgets from 10^{-2} to 10^{-6}, with no test case exceeding its specified budget in observed overflow frequency, demonstrating its robustness in settings well aligned with the theory's IID token assumption. At the same time, we also identify dependence-related temporal effects in prefill and marginal drift as causes of potential undersizing, and show that traffic mixing can reduce the severity of the former while sliding-window recalibration can effectively mitigate the impact of the latter.

Finally, we implement MarCap with the kernel `megamoe_replay`, which processes assignments that exceed the chosen capacity by reusing the same preallocated scratch workspace in successive rounds (Figure 1(b)). The smaller receive-buffer reservation computed using MarCap based on a user-specified risk budget reduces the capacity-dependent portion of operator scratch. The goal is to examine whether the kernel combined with MarCap would deliver the desired scratch reduction while approximately retaining the same latency level as the same kernel with worst-case reservation, also known as *full reservation*, when the risk budget is reasonably small, by avoiding multi-round replay in most cases. Indeed, we observed that in the EP4 execution of our kernel, with experts placed using LPT and the overflow-risk budget set to 0.01, we can achieve a 44.1–71.4% scratch workspace reduction across the tested configurations, and for the theory aligned workloads the latency is comparable to the full reservation, with median absolute differences in mean latency of 0.36% across the tested decode configurations and 0.36% across the tested shuffled-prefill configurations. Furthermore, the corresponding median absolute differences in p50 and p95 latency are 0.30% and 0.62% for the tested decode configurations, and 0.11% and 0.83% for the tested shuffled-prefill configurations, respectively. For those less theory-aligned settings, we do observe higher latency due to overflow and additional replay rounds, and larger batches can help mitigate this penalty.

Our contributions in this paper can be summarized as follows. First, we develop a mathematical theory for marginal-based risk-bounded receive-buffer provisioning in MoE expert parallelization under the IID token assumption, together with an efficient algorithm based on an exact LP reduction and a lower-hull construction. Second, we evaluate the theory's practical scope by analyzing routing records from different LLMs and identify the possible reasons for undersizing, and study how traffic mixing and sliding-window recalibration can mitigate these issues in practice. Third, we integrate the theory with a custom fixed workspace lossless replay kernel and validate its effectiveness in EP4 experiments on accelerator hardware.

2 From routing marginals to risk-bounded receiving-buffer capacity

2.1 Single-token counts and the batch-load bound

In conventional MoE dispatching without using expert duplication or token deduplication, the process can be formalized as below: there are E routing experts on EP cards, where each token selects a set S of exactly k distinct experts. The selection probabilities satisfy $q_i = \mathbb{P}(i \in S)$, $q_i \in [0, 1]$, and $\sum_i q_i = k$. Let $\mathcal{C}_c$ be the experts on card c, and we define $Z_c = |S \cap \mathcal{C}_c|$ as the number of expert assignments that one token contributes to that card, and we have $\mu_c = \mathbb{E}Z_c = \sum_{i \in \mathcal{C}_c} q_i$ to be the expected number of assignments on that card. If the card holds m experts, we must have

$$\max\{0, k - (E - m)\} \le Z_c \le \min\{k, m\}, \tag{1}$$

by pigeonhole principle. The local number of tokens to be dispatched on each card is bs, so in total we have $B = bs \cdot \text{EP}$ tokens, and after dispatching total load on a card is $L_c = \sum_{t=1}^{B} Z_c^{(t)}$. We denote the receive buffer capacity with W measured in assignment rows. Full reservation has

$W_{\max} = B \max_c \min(k, |\mathcal{C}_c|)$, while a heuristic capacity factor α reserves approximately $\frac{\alpha Bk}{\mathrm{EP}}$ assignments after rounding, where $\frac{Bk}{\mathrm{EP}}$ is simply the expected card load when the token expert choices are uniformly random and each rank has an equal number of experts.

As we mentioned above, we can not recover the distribution of Z_c from the marginals alone, where the actual top-k joint routing distribution is needed. Thus, we need to consider all top-k routing distributions with marginals q, denoted by $\mathcal{P}(q)$, and compute

$$M_{\mathrm{ext},c}(\theta) = \max_{P \in \mathcal{P}(q)} \mathbb{E}_P e^{\theta Z_c}, \qquad \theta > 0. \tag{2}$$

We use the MGF as it puts more emphasis on the tail behaviors and it turns a sum of independent token loads into a product of their MGFs under the IID token assumption, and Markov's inequality gives for $\theta > 0$:

$$\mathbb{P}(L_c > W) = \mathbb{P}(e^{\theta L_c} > e^{\theta W}) \le e^{-\theta W} \mathbb{E} e^{\theta L_c} \le e^{-\theta W} \big(M_{\mathrm{ext},c}(\theta)\big)^B. \tag{3}$$

For a selected overflow risk $\varepsilon \in (0, 1)$, allocate overflow budget $\frac{\varepsilon}{\mathrm{EP}}$ to each card and use the union bound. Solving (3) for a minimized universal W across all ranks gives

$$W_{\mathrm{mar}}(\varepsilon) = \left\lceil \max_c \inf_{\theta > 0} \frac{B \log M_{\mathrm{ext},c}(\theta) + \log(\mathrm{EP}/\varepsilon)}{\theta} \right\rceil \Longrightarrow \mathbb{P}\Big(\max_c L_c > W_{\mathrm{mar}}\Big) \le \varepsilon. \tag{4}$$

Note equation (4) is a strictly guaranteed conservative capacity bound when token routes are IID and the given q matches the true expert marginals.

2.2 An exact Linear Program with at most $k + 1$ variables

Brute force computation for (2) is impossible as it would require $\binom{E}{k}$ variables. For $E = 256$ and $k = 8$, this is about 4.1×10^{14}. However, the question can be greatly simplified as the per token card load Z_c doesn't need to distinguish which expert is selected. For a fixed card let $p_r = \mathbb{P}(Z_c = r)$, for $r = 0, \ldots, k$. All expert selections that produce count r have the same MGF weight $e^{\theta r}$, so their probabilities can be grouped.

The question then comes from how to identify such a probability vector p that lives in the simplex that is compatible with the given marginals q, and we have the following theorem.

Theorem 1 (Feasibility condition of per-card count probabilities)**.** *A vector $p = (p_0, \ldots, p_k)$ is realizable by a top-k routing distribution with given expert marginals if and only if it satisfies the following constraints. Consequently, the maximum MGF is exactly the value of this linear program:*

$$\begin{aligned}
&M_{\mathrm{ext},c}(\theta) = \max_p \textstyle\sum_{r=0}^k e^{\theta r} p_r, \quad \textit{subject to} \quad p_r \ge 0, \quad \textstyle\sum_{r=0}^k p_r = 1, \quad \textstyle\sum_{r=0}^k r p_r = \sum_{j=1}^m q_j, \\
&\textstyle\sum_{j=1}^{\ell} q_j \le \sum_{r=0}^k \min(\ell, r) p_r, \quad \ell = 1, \ldots, m, \\
&\textstyle\sum_{j=1}^{\ell} g_j \le \sum_{r=0}^k \min(\ell, k - r) p_r, \quad \ell = 1, \ldots, E - m,
\end{aligned} \tag{5}$$

where q_j's are marginals for experts on card c in descending order and g_j's are marginals for experts outside card c in descending order.

Sketch of proof. Necessity can be justified using a counting argument as p_r's must be large enough to guarantee the possible assignment probability to expert i is at least q_i, and the same argument can be made for experts that are not on card c. Sufficiency can be justified using the max-flow/min-cut conditions from graph theory (Gale, 1957). The resulting flows yield conditional marginals realizable by fixed-size subset distributions, whose combination gives the required routing law. The full proof is included in appendix A. □

Thus, the reduction preserves the original optimization exactly, but it requires solving a small LP for each θ. The next step will remove such redundancy.

2.3 Universal MGF for each card

We can define the expected excess count for a card above an integer threshold s by

$$C_s = \mathbb{E}[(Z_c - s)_+] = \sum_{r=0}^k p_r \max(r - s, 0), \qquad s = 0, \ldots, k. \tag{6}$$

Here $(x)_+ = \max(x, 0)$, so $(Z_c - s)_+$ is a shifted ReLU. Let prefix sums $Q_s = \sum_{j=1}^{\min(s,m)} q_j$ and $G_s = \sum_{j=1}^{\min(s,E-m)} g_j$, with empty sums zero. The identities $\min(s, Z_c) = Z_c - (Z_c - s)_+$ and $\min(k - s, k - Z_c) = k - s - (Z_c - s)_+$ then turn the two constraint families above into

$$C_s \le U_s = \min\{\mu_c - Q_s,\ k - s - G_{k-s}\}. \tag{7}$$

Moreover, we have boundary conditions $C_0 = \mu_c$, $C_k = 0$, and $C_{s-1} - C_s = \mathbb{P}(Z_c \ge s)$. Thus, the consecutive differences are decreasing probabilities, so C_s is a decreasing convex sequence with slopes between -1 and 0.

Theorem 2 (θ Universal MGF Maximizer)**.** *The pointwise largest convex sequence below the bounds U_s represents a feasible count distribution $p^\star$, and it uniquely maximizes (5) for every positive θ.*

Sketch of Proof. Consider the largest convex sequence H below U, and by comparison with any feasible C we can immediately conclude that H also satisfies the boundary and consecutive difference conditions and is therefore feasible. For fixed μ_c and θ, the MGF can be expressed as a constant plus a strictly positive weighted sum of C_s. Therefore, the universal maximizing C, i.e. H, shall give the MGF-maximizing distribution $p^\star$ independent of positive θ. More details can be found in appendix A.2. Once $p^\star$ is determined, equation (4) can be solved efficiently using standard root-finding algorithms (Appendix A.3). □

2.4 Efficient algorithm speedup

While the maximizer can be solved with a single linear program for each card, a lower convex hull method is the most efficient one as it removes the need for an LP solver. The desired H can be reconstructed with an $O(k)$ scan once the dominating U is computed. The detailed algorithm is included in appendix A.2. We compare the planning time (Table 1) for each of the three algorithms on the same CPU hardware with prescribed routing marginals we obtained from two LLM model deployments and check the consistency of the numerical results (Appendix B), and the methods produce identical integer capacities, with only negligible differences in the continuous capacities, while the hull method is $94.4\times$ and $109.9\times$ faster than the naive one LP per θ version and $8.50\times$ and $5.46\times$ faster than the one LP per card model, based on median paired speedups.

Table 1: Capacity planning time per layer: median seconds over five paired repetitions. Median paired speedup measured against per-parameter LP. All methods use the same marginals, placement, batch sizes and risk budget.

Model	LP at each θ	One LP/card	Hull	Speedup
Qwen	8.258	0.735	0.087	$94.4\times$
DeepSeek	2.493	0.124	0.023	$109.9\times$

3 Routing experiments

3.1 Experimental setup and key metrics

We use routing records from Qwen3.6-35B-A3B and DeepSeek-V2-Lite-Chat (Qwen Team, 2026; DeepSeek-AI, 2024; Zhao et al., 2024), covering 2.737 billion tokens from WildChat-1M requests. We retain each token's complete expert selections for this offline evaluation. MarCap uses only the marginals computed from the calibration data to choose capacity, while the held-out per-token records allow us to reconstruct batch loads and measure overflow. We first run three controlled tests to help determine the theory's practical scope: contiguous prefill keeps tokens from the same request in order, while shuffled prefill randomly permutes the same tokens; decode samples recorded token routes independently with replacement, using 10000 batches per configuration. They should be regarded as stress tests rather than faithful representations of real world serving because they use fully populated batches, while real serving may have fewer active tokens with padded positions excluded from routing, and a Poisson arrival model is more appropriate (Section 3.4). Nevertheless, these tests are still valuable because shuffled prefill closely aligns with the IID token assumption and comparison with contiguous prefill helps reveal the effect of inter-token dependence within microbatches, while the decode test approximates the one token per request setting of conventional decoding without aggressive multi-token speculative decoding.

We use 25% of the requests to compute expert marginals and place experts using LPT, and use the remaining 75% for evaluation. The marginals and placement are fixed during evaluation, and we set the risk budget to 0.01. These tests preserve the expert choices within each token. Table 2 shows the tested configurations. Calibration details and overflow results are in Appendix G.

Table 2: Relevant model parameters and configurations covered. EP and bs values use powers of two.

Model	E	k	Layers	EP	Local bs
Qwen	256	8	0, 19, 39	2–256	1–8192
DeepSeek	64	6	3, 14, 26	2–64	1–131072

We define $W_{\mathrm{req}}(\varepsilon)$ as the smallest receive-buffer capacity that keeps the observed overflow rate at most ε. We report the compression rate η, the capacity factor (CF) gap and the observed overflow probability $\widehat{o}$ as the key metrics over n evaluated batches. Note that $\Delta\mathrm{CF} \leq 0$ means the observed overflow frequency is at most the risk budget.

$$\eta = 1 - \frac{W_{\mathrm{mar}}}{W_{\max}}, \qquad \Delta\mathrm{CF} = \frac{W_{\mathrm{req}} - W_{\mathrm{mar}}}{Bk/\mathrm{EP}}, \qquad \widehat{o} = \frac{1}{n}\sum\nolimits_j \mathbf{1}_{\{\max_c L_c^{(j)} > W_{\mathrm{mar}}\}}. \tag{8}$$

3.2 Explaining ΔCF: traffic mixing, temporal dependence, and marginal drift

Table 3: Receive-buffer compression rate (%) with MarCap Capacity CF in parentheses.

Model	Local bs	EP4	EP8	EP16	EP32	EP64
Qwen L19	16	54.88 (1.805)	75.59 (1.953)	87.30 (2.031)	93.99 (1.922)	93.77 (1.992)
Qwen L19	128	68.16 (1.273)	83.56 (1.315)	91.64 (1.338)	95.94 (1.298)	94.81 (1.662)
Qwen L19	512	71.61 (1.135)	85.56 (1.155)	92.72 (1.166)	96.42 (1.145)	95.07 (1.577)
Qwen L19	2048	73.31 (1.068)	86.54 (1.077)	93.23 (1.083)	96.61 (1.086)	95.20 (1.536)
Qwen L19	8192	74.14 (1.034)	87.02 (1.039)	93.48 (1.043)	96.69 (1.060)	95.26 (1.515)
DeepSeek L14	16	54.69 (1.812)	75.39 (1.969)	82.91 (1.823)	84.86 (1.615)	81.74 (1.948)
DeepSeek L14	128	68.03 (1.279)	83.33 (1.333)	87.93 (1.288)	88.53 (1.224)	84.83 (1.618)
DeepSeek L14	512	71.50 (1.140)	85.36 (1.172)	89.21 (1.151)	89.43 (1.128)	85.64 (1.532)
DeepSeek L14	2048	73.20 (1.072)	86.34 (1.093)	89.83 (1.084)	89.87 (1.081)	86.04 (1.489)
DeepSeek L14	8192	74.04 (1.039)	86.82 (1.054)	90.14 (1.052)	90.09 (1.057)	86.24 (1.468)

Table 3 shows two of the metrics for selected decode configurations. We can see that the calculated capacity factor varies across configurations, making it difficult to choose a single fixed factor that both meets the risk budget and avoids unnecessary reservation. Figure 2 shows that while ΔCF generally remains at or below the x-axis for shuffled prefill or decode, which supports the effectiveness of MarCap in these two settings, contiguous prefill has much larger ΔCF (Figures 14–19). We consider the reason to be that consecutive tokens from the same request can favor the same experts, creating temporal inter-token dependence that MarCap's IID model does not account for. Appendix C attempts to address this issue by allowing arbitrary dependence among tokens from the same request while assuming that requests are independent and each contributes a bounded number of tokens to a batch. Moreover, ΔCF tends to decrease at larger batch sizes, which we hypothesize can be explained by more thorough mixing of requests as larger batches naturally include tokens from more requests. For Qwen L19/EP256, the gap falls from 23.625 at $bs = 2$ to 1.159 at $bs = 8192$. Furthermore, such mixing at a finer token granularity could also help explain why the other two settings align well with the theory.

We need to also point out that for the better aligned settings even if ΔCF turns out to be small, the risk budget violation could be significant. For Qwen L39/EP256 decode, the CF gap is only 0.029434, yet the observed overflow frequency is 95.54%. This failure can be attributed to the marginal difference between the calibration and evaluation sets. The marginal shift has ℓ_1 norm 0.04281. At large batch sizes, loads concentrate around the shifted mean, so even a small capacity deficit can affect most batches.

3.3 Robustness across risk budgets

To verify robustness of MarCap with different risk budgets, we test risk budgets of 10^{-2}, 10^{-4} and 10^{-6} while keeping other variables fixed. We use two batch/EP settings per model and all three layers

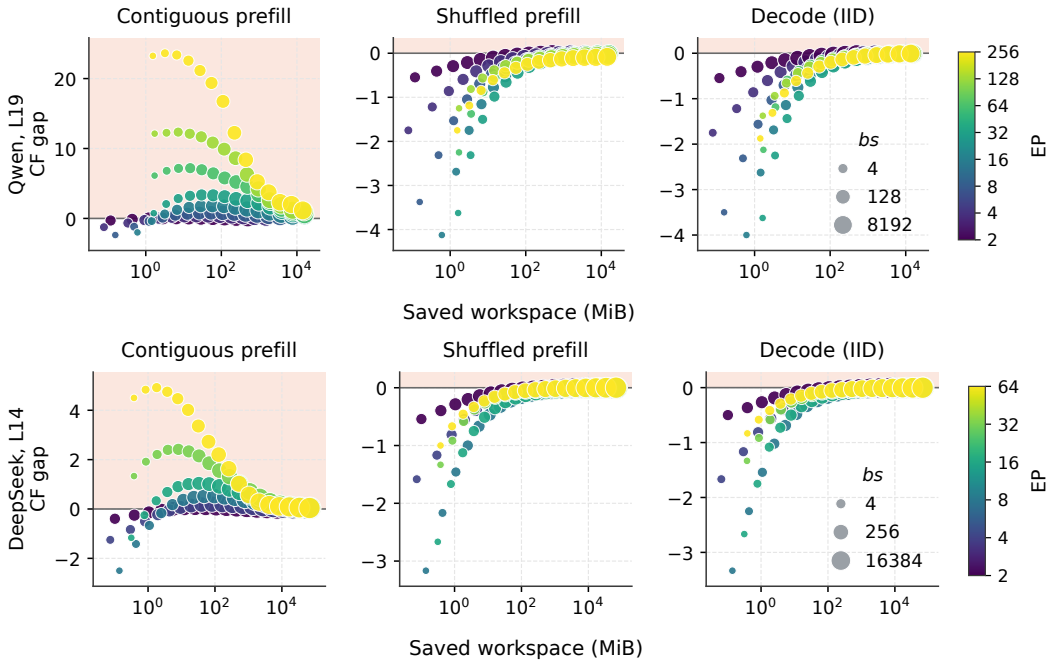


Figure 2: **Frozen-calibration stress test results.** CF gap versus estimated capacity-dependent workspace savings in MiB across three settings and two models. Color denotes EP and marker size denotes local bs. Positive CF gaps are shaded. Details in Appendix G.3.

for shuffled prefill and decode, and observe no over-budget overflow. Tightening the budget from 10^{-2} to 10^{-6} increases capacity by 4.40–16.51%, while still saving 84.24–94.84% relative to full reservation (Table 4). Each decode configuration uses ten million sampled batches. More details for the robustness experiment are included in Appendix H.

Table 4: Capacity increase and savings across risk budgets. Each row summarizes 12 settings with no observed overflow; the increase is relative to $\varepsilon = 0.01$.

Model	ε	Capacity increase (%)	Compression (%)	Max. ΔCF
Qwen	10^{-2}	0.00	90.75–95.53	−0.0938
Qwen	10^{-4}	2.34–8.91	90.47–95.15	−0.1045
Qwen	10^{-6}	4.40–16.51	90.23–94.84	−0.1084
DeepSeek	10^{-2}	0.00	85.30–88.79	−0.1090
DeepSeek	10^{-4}	3.89–4.41	84.72–88.29	−0.1296
DeepSeek	10^{-6}	7.15–8.11	84.24–87.88	−0.1452

3.4 Sliding-window recalibration

To mitigate marginal drift which could affect the provisioning guarantee in real deployments, we adopt a sliding window method to update the marginals from completed routing history and compute the capacity before observing the next window. To leverage the law of large numbers, we use a large routing history to reduce marginal estimation error: four windows of 524288 tokens for prefill and one window of 7340032 tokens for decode. We compare it with frozen calibration using the same placement and future traffic. Note that for changing capacities, CF gaps are computed from pooled load minus capacity residuals so the sign relationship is preserved (Appendix I.1).

We begin with a Poisson process model for request arrivals to mimic real-life serving situations. Figure 3 plots prefill at an offered load ratio of 0.75 and decode at 0.95, with each active decode request contributing one token per step. The sliding window method eliminates the decode overflow observed with frozen marginals and capacities, while narrowing the CF gap in the plotted prefill

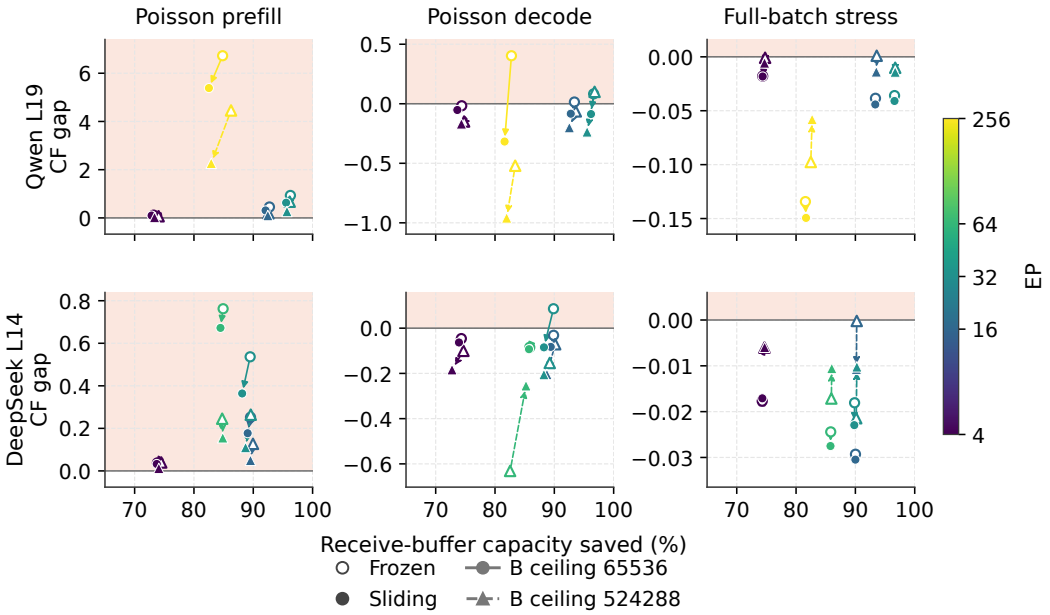


Figure 3: **Sliding recalibration effect on CF gaps.** Frozen (open) to Sliding (filled) on matched future steps for Qwen L19 and DeepSeek L14 at selected EP values and both batch limits. Details in Appendix I.3.

configurations. For DeepSeek L14/EP32 at global batch limit 65536, overflow falls from 82.89% to zero while retaining 88.25% capacity savings.

Because Poisson arrivals do not always fill the batch, we also stress test full batches for decode. We initialize the first batch of requests at different decode ages to avoid the artificial synchronization caused by starting all requests at the same age. The sliding window method also gives zero observed overflow in all such tested configurations, with capacity savings of 49.29–96.71% (Appendix I.2).

4 Kernel Implementation and Hardware Experiments

We implement MarCap in `megamoe_replay`, which processes assignments in successive rounds when the receive-buffer capacity W is exceeded (Figure 1(b)). Each round dispatches at most W assignments per destination, performs two GEMMs with activation, and writes the results back to their source slots through combine. The scratch workspace can then be reused for the next round and unpermute applies the routing weights and combines the results after all assignments have been processed, so the kernel can execute lossless MoE computation with a smaller fixed workspace (Appendices J.2 and J.3).

Hardware setup. We run EP4 experiments on Ascend 950PR using Qwen L19 records, with $E = 256$, $k = 8$, hidden width 2048, and intermediate width 512. We use constructed BF16 activations and MXFP8 E4M3 weights with E8M0 scales. Experts are placed using LPT and the risk budget is 0.01. We compare the kernel latency under MarCap with full reservation and fixed capacity factors of 1 and 1.25, excluding planning time. Each configuration has five runs of 1000 calls timed by the maximum rank-local event duration. All checked outputs exactly match the expected results for the constructed inputs.

Scratch savings. For packed prefill at local batch sizes 512, 2048 and 8192, total per-rank scratch falls from 172.52 to 96.43 MiB, 491.27 to 179.67 MiB, and 1766.27 to 505.60 MiB, respectively, giving reductions of 44.1%, 63.4% and 71.4%. In particular, at $bs = 512$, the capacity-dependent part falls from 106.25 to 30.16 MiB while 66.27 MiB of fixed allocation overhead remains fixed (Table 23), and it explains why total scratch savings are smaller than receive-buffer capacity savings.

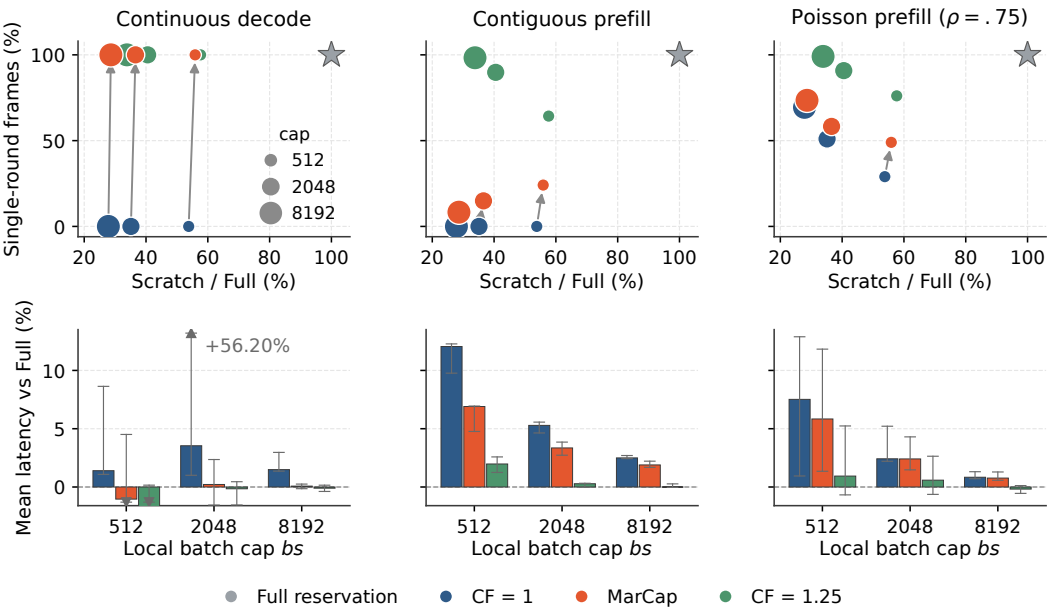


Figure 4: **Scratch savings and replay overhead.** Top: total scratch relative to full reservation versus single-round frequency; marker size denotes batch cap. Bottom: paired mean-latency changes from full reservation. Whiskers show descriptive repetition/panel ranges, not confidence intervals; triangles mark clipped ends. Detailed hardware results are in Appendix J.

Latency. Decode and shuffled prefill have no observed overflow for MarCap capacity. The median absolute differences in mean latency from full reservation are 0.36% across decode configurations and 0.36% across shuffled-prefill configurations. For p50 and p95, the differences are 0.30% and 0.62% for decode, and 0.11% and 0.83% for shuffled prefill, respectively. In comparison, for continuous decode at $bs = 8192$, capacity factor 1 always requires replay, while MarCap finishes every batch in one round, using extra scratch over CF1.0 equal to only 0.8% of the full-reservation scratch.

For contiguous prefill, overflow causes additional rounds and contributes to higher latency. The mean-latency increase falls from 6.90% to 1.89% as the batch size grows from 512 to 8192. Shuffling the same tokens reduces MarCap's mean latency by 10.72–15.46%. Full reservation also benefits from shuffling, so this improvement is not due to avoiding replay alone as reduced straggling could also contribute.

5 Related Work and Conclusion

MegaBlocks enables dropless block-sparse computation, while dynamic gating uses variable token groups, and both exchange actual counts before dispatch (Gale et al., 2023; Huang et al., 2024). Tutel and DeepSpeed-MoE also support dropless execution with capacity adapted to realized expert loads (Hwang et al., 2023; Rajbhandari et al., 2022). Specific vLLM and SGLang backends use chunking or workspace reuse for intermediate computation buffers (Kwon et al., 2023; Zheng et al., 2024). DeepEP and NCCL-EP optimize dispatch and combine (DeepSeek-AI, 2026a; Goldman et al., 2026). Capacity-aware inference reduces stragglers by dropping tokens or expanding expert candidates (He et al., 2026).

MoonEP and UltraEP balance realized loads through online expert replication and assignment redistribution (Chen et al., 2026; Wei et al., 2026). MarCap instead keeps placement fixed and computes capacity from marginals and a risk budget under the IID token assumption, then handles overflow through lossless replay in the same workspace. Its exact reduction and lower-hull computation make provisioning efficient, while traffic mixing and marginal recalibration mitigate undersizing. The same planning idea could guide the GPU capacity of elastic receive buffers whose excess load uses CPU memory (Appendix J.1).

AI USE STATEMENT

Generative AI assisted writing refinement, literature searches, mathematical theory verification, numerical implementation, data processing and analysis, and figure generation. Numerical claims were checked against the experimental records.

ETHICS STATEMENT

The study uses WildChat-derived routing data. The accompanying artifact contains routing statistics and operator measurements; conversation text and user identifiers are absent. Access to the source dataset remains subject to its terms.

REPRODUCIBILITY STATEMENT

Appendix A proves the core results, and Appendices B, G and J describe the experiments. The supplementary material provides the capacity-planning implementation, numerical verification, routing statistics, per-invocation hardware timings, and scripts for re-analysing the saved results and reproducing the main experimental plots and all appendix figures. The captured expert-ID routing records are available at https://huggingface.co/datasets/m38ecaaed/marcap-routing.

REFERENCES

Yutian Chen, Cong Li, Yucheng Wang, and Ming Wei. MoonEP: A perfectly balanced expert parallelism library via dynamic redundant experts. Official software repository, 2026. URL https://github.com/MoonshotAI/MoonEP. Commit 33327eb9c4a8; accessed September 22, 2026.

DeepSeek-AI. DeepSeek-V2-Lite-Chat. Official model card, 2024. URL https://huggingface.co/deepseek-ai/DeepSeek-V2-Lite-Chat.

DeepSeek-AI. DeepEP: An efficient expert-parallel communication library. Official software repository, 2026a. URL https://github.com/deepseek-ai/DeepEP. Accessed September 5, 2026.

DeepSeek-AI. Expert parallelism load balancer (EPLB). Official software repository, 2026b. URL https://github.com/deepseek-ai/EPLB. Accessed September 5, 2026.

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. URL https://www.jmlr.org/papers/v23/21-0998.html.

David Gale. A theorem on flows in networks. *Pacific Journal of Mathematics*, 7(2):1073–1082, 1957. URL https://msp.org/pjm/1957/7-2/pjm-v7-n2-p04-p.pdf.

Trevor Gale, Deepak Narayanan, Cliff Young, and Matei Zaharia. MegaBlocks: Efficient sparse training with mixture-of-experts. In *Proceedings of Machine Learning and Systems*, volume 5, 2023. URL https://proceedings.mlsys.org/paper_files/paper/2023/hash/5a54f79333768effe7e8927bcccffe40-Abstract-mlsys2023.html.

Seokjin Go and Divya Mahajan. MoETuner: Optimized mixture of expert serving with balanced expert placement and token routing. *arXiv preprint arXiv:2502.06643*, 2025. doi: 10.48550/arXiv.2502.06643. URL https://arxiv.org/abs/2502.06643.

Amos Goldman, Nimrod Boker, Maayan Sheraizin, Nimrod Admoni, Artem Polyakov, Subhadeep Bhattacharya, Fan Yu, Kai Sun, Georgios Theodorakis, Hsin-Chun Yin, Peter-Jan Gootzen, Aamir Shafi, Assaf Ravid, Salvatore Di Girolamo, James Dinan, Xiaofan Li, Manjunath Gorentla Venkata, and Gil Bloch. NCCL EP: Towards a unified expert parallel communication API for NCCL. *arXiv preprint arXiv:2603.13606*, 2026. doi: 10.48550/arXiv.2603.13606. URL https://arxiv.org/abs/2603.13606v3.

Gaston H. Gonnet. Expected length of the longest probe sequence in hash code searching. *Journal of the ACM*, 28(2):289–304, 1981. doi: 10.1145/322248.322254. URL https://doi.org/10.1145/322248.322254.

Shwai He, Weilin Cai, Jiayi Huang, and Ang Li. Capacity-aware inference: Mitigating the straggler effect in mixture of experts. In *International Conference on Learning Representations*, 2026. URL https://proceedings.iclr.cc/paper_files/paper/2026/hash/94e845868a9ace4bc239d0c529d32f4c-Abstract-Conference.html.

Haiyang Huang, Newsha Ardalani, Anna Sun, Liu Ke, Hsien-Hsin S. Lee, Shruti Bhosale, Carole-Jean Wu, and Benjamin Lee. Toward efficient inference for mixture of experts. In *Advances in Neural Information Processing Systems*, volume 37, 2024. doi: 10.52202/079017-2670. URL https://proceedings.neurips.cc/paper_files/paper/2024/hash/98bf3b8505c611ac21055dd9d355c66e-Abstract-Conference.html.

Changho Hwang, Wei Cui, Yifan Xiong, Ziyue Yang, Ze Liu, Han Hu, Zilong Wang, Rafael Salas, Jithin Jose, Prabhat Ram, HoYuen Chau, Peng Cheng, Fan Yang, Mao Yang, and Yongqiang Xiong. Tutel: Adaptive mixture-of-experts at scale. In *Proceedings of Machine Learning and Systems*, volume 5, 2023. URL https://proceedings.mlsys.org/paper_files/paper/2023/hash/5616d34cf8ff73942cfd5aa922842556-Abstract-mlsys2023.html.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with PagedAttention. In *Proceedings of the ACM SIGOPS 29th Symposium on Operating Systems Principles*, 2023. URL https://arxiv.org/abs/2309.06180.

Dmitry Lepikhin, HyoukJoong Lee, Yuanzhong Xu, Dehao Chen, Orhan Firat, Yanping Huang, Maxim Krikun, Noam Shazeer, and Zhifeng Chen. GShard: Scaling giant models with conditional computation and automatic sharding. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=qrwe7XHTmYb.

NVIDIA. NCCL device API: GIN. NCCL 2.32.3 documentation, 2026. URL https://docs.nvidia.com/deeplearning/nccl/archives/nccl_2323/user-guide/docs/api/device_gin.html. Accessed September 26, 2026.

Giovanni Puccetti and Ludger Rüschendorf. Sharp bounds for sums of dependent risks. *Journal of Applied Probability*, 50(1):42–53, 2013. doi: 10.1239/jap/1363784423. URL https://doi.org/10.1239/jap/1363784423.

Zihan Qiu, Zeyu Huang, Bo Zheng, Kaiyue Wen, Zekun Wang, Rui Men, Ivan Titov, Dayiheng Liu, Jingren Zhou, and Junyang Lin. Demons in the detail: On implementing load balancing loss for training specialized mixture-of-expert models. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 5005–5018, Vienna, Austria, July 2025. Association for Computational Linguistics. doi: 10.18653/v1/2025.acl-long.249. URL https://aclanthology.org/2025.acl-long.249/.

Qwen Team. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025. doi: 10.48550/arXiv.2505.09388. URL https://arxiv.org/abs/2505.09388.

Qwen Team. Qwen3.6-35B-A3B: Agentic coding power, now open to all, April 2026. URL https://qwen.ai/blog?id=qwen3.6-35b-a3b.

Martin Raab and Angelika Steger. "Balls into bins" — a simple and tight analysis. In *Randomization and Approximation Techniques in Computer Science*, volume 1518 of *Lecture Notes in Computer Science*, pp. 159–170. Springer Berlin Heidelberg, 1998. doi: 10.1007/3-540-49543-6_13. URL https://doi.org/10.1007/3-540-49543-6_13.

Richard Rado. An inequality. *Journal of the London Mathematical Society*, s1-27(1):1–6, 1952. doi: 10.1112/jlms/s1-27.1.1. URL https://doi.org/10.1112/jlms/s1-27.1.1.

Samyam Rajbhandari, Conglong Li, Zhewei Yao, Minjia Zhang, Reza Yazdani Aminabadi, Ammar Ahmad Awan, Jeff Rasley, and Yuxiong He. DeepSpeed-MoE: Advancing mixture-of-experts inference and training to power next-generation AI scale. In *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pp. 18332–18346. PMLR, 2022. URL https://proceedings.mlr.press/v162/rajbhandari22a.html.

Moshe Shaked and J. George Shanthikumar. *Stochastic Orders*. Springer, 2007. doi: 10.1007/978-0-387-34675-5. URL https://link.springer.com/book/10.1007/978-0-387-34675-5.

Yifan Sun, Zhexiang Zhang, Jiantong Jiang, Gholamreza Haffari, Minxian Xu, Feng Liu, Rajkumar Buyya, and Adel N. Toosi. Coordinated scheduling for MoE LLM serving. *arXiv preprint arXiv:2606.15177*, 2026. doi: 10.48550/arXiv.2606.15177. URL https://arxiv.org/abs/2606.15177.

Xinming Wei, Chao Jin, Tuo Dai, Yinmin Zhong, Shan Yu, Chengxu Yang, Bingyang Wu, Zili Zhang, Jing Mai, Qianchao Zhu, Zhouyang Li, Yuliang Liu, and Guojie Luo. UltraEP: Unleash MoE training and inference on rack-scale nodes with near-optimal load balancing. *arXiv preprint arXiv:2606.04101*, 2026. doi: 10.48550/arXiv.2606.04101. URL https://arxiv.org/abs/2606.04101v3.

Wenting Zhao, Xiang Ren, Jack Hessel, Claire Cardie, Yejin Choi, and Yuntian Deng. WildChat: 1M ChatGPT interaction logs in the wild. In *International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=Bl8u7ZRlbM.

Lianmin Zheng, Liangsheng Yin, Zhiqiang Xie, Chuyue Sun, Jeff Huang, Cody Hao Yu, Shiyi Cao, Christos Kozyrakis, Ion Stoica, Joseph E. Gonzalez, Clark Barrett, and Ying Sheng. SGLang: Efficient execution of structured language model programs. In *Advances in Neural Information Processing Systems*, volume 37, 2024. doi: 10.52202/079017-2000. URL https://proceedings.neurips.cc/paper_files/paper/2024/hash/724be4472168f31ba1c9ac630f15dec8-Abstract-Conference.html.

Yinmin Zhong, Shengyu Liu, Junda Chen, Jianbo Hu, Yibo Zhu, Xuanzhe Liu, Xin Jin, and Hao Zhang. DistServe: Disaggregating prefill and decoding for goodput-optimized large language model serving. In *18th USENIX Symposium on Operating Systems Design and Implementation (OSDI 24)*, pp. 193–210, Santa Clara, CA, July 2024. USENIX Association. URL https://www.usenix.org/conference/osdi24/presentation/zhong-yinmin.

APPENDIX CONTENTS

A Proofs of the main results **15**

A.1 Exact characterization of feasible count probabilities 15

A.2 Universal MGF maximizer and lower convex hull construction 16

A.3 Risk-bounded capacity and scalar minimization 19

B Numerical agreement and CPU planning cost **21**

B.1 Full LP versus reduced LP 21

B.2 Numerical agreement of the three methods 21

C A batch-load bound for independent blocks **22**

D Token-deduplicated receive buffers **23**

D.1 Capacity from expert marginals 23

D.2 Experimental setup and metrics 24

D.3 Results 25

E Replicated experts and expert-parallel load balancing **27**

E.1 Count feasibility with physical routing marginals 27

E.2 Source-dependent dispatch 28

E.3 Experimental setup and metrics 28

E.4 Results 29

F Experimental setup for Figure 1(a) **31**

G Routing experiments with 25% calibration **31**

G.1 Calibration and batch construction 31

G.2 Overflow in shuffled prefill and decode 32

G.3 Receive-buffer compression and CF gaps across layers 33

G.4 Quantifying temporal effects 35

H Robustness across risk budgets **38**

I Sliding-window recalibration **39**

I.1 Prefill and decode with Poisson arrivals 39

I.2 Decode with Poisson arrivals and full batches 41

I.3 Complete results for Figure 3 44

J MegaMoe Replay: implementation and hardware experiments **51**

J.1 Capacity and buffers in existing MoE systems 51

J.2 Replay with bounded workspace 51

J.3 Workspace reuse and output accumulation 52
J.4 Workloads, timing and correctness 52
J.5 Paired comparison against full reservation 53
J.6 Latency mechanisms 56

A PROOFS OF THE MAIN RESULTS

Notations. In this appendix, $1 \le k \le E$ are integers, the marginals satisfy $q_i \in [0,1]$ and $\sum_i q_i = k$, and we have a fixed expert placement. We use the inside Q_s and outside G_s marginal sums defined in Section 2.3. The first two subsections concern a single token and the IID token assumption is used for the batch-load bound in Appendix A.3.

A.1 EXACT CHARACTERIZATION OF FEASIBLE COUNT PROBABILITIES

We first show that the assumed conditions on the expert marginals are sufficient for a joint routing distribution to exist. We will then apply this result to the conditional marginals obtained from the flow construction.

Lemma A.1 (Existence of a joint routing distribution)**.** *Let $E \ge 0$ and $0 \le k \le E$ be integers. For any $q \in [0,1]^E$ satisfying $\sum_i q_i = k$, there exists a probability distribution over subsets $S \subseteq \{1,\dots,E\}$ such that $|S| = k$ almost surely and $\mathbb{P}(i \in S) = q_i$ for every i, i.e. $\mathcal{P}(q)$ is nonempty.*

Proof. Consider the polytope $\{q \in [0,1]^E : \sum_i q_i = k\}$. Any point with a fractional coordinate must have at least two such coordinates because its coordinate sum is an integer. Increasing one and decreasing the other by the same small amount and the opposite operation will create two distinct points in the polytope, with their midpoint being the original point. Therefore, every vertex can only be a vector with only zeros and ones with exactly k ones. Because a bounded polytope is the convex hull of its vertices, q is then a convex combination of these vectors. Each vertex specifies a set S of k experts. Choosing these sets with the corresponding coefficients as probabilities yields the desired joint routing distribution. □

We use $\mathcal{S}_k$ to denote the collection of all k-element subsets of $\{1,\dots,E\}$. A routing distribution is specified by probabilities $\pi_S \ge 0$ with $\sum_{S\in\mathcal{S}_k} \pi_S = 1$ and $\sum_{S\ni i} \pi_S = q_i$, and the count probabilities are then

$$p_r = \sum_{S:|S\cap\mathcal{C}_c|=r} \pi_S. \tag{9}$$

Proof of Theorem 1. We first prove necessity. For any group T of inside experts, a routing result with assignment to card count r chooses at most $\min(|T|, r)$ experts from T. By taking expectation, we have

$$\sum_{i\in T} q_i \le \sum_{r=0}^{k} \min(|T|, r) p_r. \tag{10}$$

For each group size ℓ, the largest possible left-hand side is the sum of the ℓ largest inside marginals, which gives the inside constraints in (5). Similar arguments produce the outside constraints where $k-r$ experts are selected. Nonnegativity, normalization and the mean equality are direct consequences from the definition of p_r.

To show sufficiency, suppose that p satisfies the constraints in (5). We seek numbers x_{ir} for $i \in \mathcal{C}_c$ that will reproduce the joint probability of selecting expert i and having count r, where they must satisfy

$$\sum_r x_{ir} = q_i, \qquad \sum_i x_{ir} = r p_r, \qquad 0 \le x_{ir} \le p_r. \tag{11}$$

We construct a network (Figure 5) with source-to-expert capacities q_i, expert-to-count-node capacities p_r, and count-node-to-sink capacities rp_r. Notice here the total capacity leaving the source and entering the sink are both μ_c. For the cut condition, fix the set T of expert nodes on the source side. Placing count node r on the source side contributes rp_r to the cut, while placing it on the sink side contributes $|T|p_r$. Minimizing over these choices gives

$$\mu_c - \sum_{i\in T} q_i + \sum_r \min(|T|, r) p_r. \tag{12}$$

The sorted inside constraints imply (10) for every T, so every cut has capacity at least μ_c. Therefore, the max-flow/min-cut theorem gives a flow of value μ_c (Gale, 1957). Every edge leaving the source

or entering the sink then carries its full capacity, and flow conservation together with the capacity bounds on the expert-to-count edges gives (11). The outside network is constructed in the same way with count-node-to-sink capacities $(k-r)p_r$ and total flow $k-\mu_c$. Its cut condition follows from the outside constraints. This argument works for real-valued marginals and probabilities.

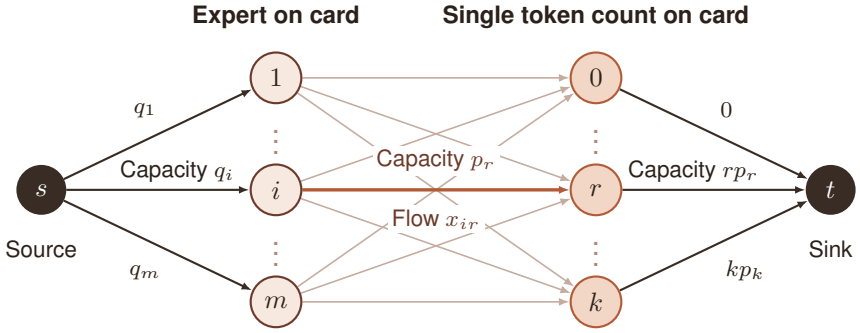


Figure 5: The flow network connects each expert on card c to all count nodes, with intermediate nodes omitted.

It remains to construct the routing distribution from the two flows. For each r with $p_r > 0$, the inside vector $u_i^{(r)} = \frac{x_{ir}}{p_r}$ has entries in $[0,1]$ and sum r. Lemma A.1 realizes it as the inclusion probabilities of an inside subset of size r. First we sample r according to p, and then sample the inside and outside subsets independently conditional on r. Their union has exactly k distinct experts and count r on card c. Moreover, for each inside expert,

$$\mathbb{P}(i \text{ is selected}) = \sum_{r:p_r>0} p_r \frac{x_{ir}}{p_r} = \sum_r x_{ir} = q_i,$$

where $x_{ir} = 0$ whenever $p_r = 0$. The same argument recovers every outside marginal. Therefore, p is realizable. Finally, because the MGF is only dependent on these count probabilities, optimizing it over the feasible vectors p is exactly equivalent to optimizing over $\mathcal{P}(q)$. □

Remark A.2 (Connection to majorization). Rado's theorem (Rado, 1952) gives another way to construct a routing distribution from feasible count probabilities. Suppose Z_c has distribution p and satisfies the bounds in (1), and set $a_j = \mathbb{P}(Z_c \geq j)$ for $j = 1, \ldots, m$. Because $\sum_{j=1}^{\ell} a_j = \sum_r \min(\ell, r)p_r$, the inside prefix constraints and mean equality imply that the inside marginal vector is majorized by a. The theorem then expresses that vector as a convex combination of permutations of a. Choosing a random permutation independent of Z_c according to this mixture and selecting its first Z_c experts then realizes both the marginals and the count distribution. Applying the same construction outside with count $k - Z_c$ and combining the two selections produces the desired routing distribution.

A.2 Universal MGF maximizer and lower convex hull construction

For simplicity we fix a card and write $Z = Z_c$ with $\mu = \mu_c$. To prove Theorem 2, we describe the same feasible count distributions using the expected excess counts $C_s = \mathbb{E}[(Z-s)_+]$, so we can compare their MGFs using the values of C_s.

Lemma A.3 (Feasibility using expected excess counts)**.** *The feasible count conditions in (5) are exactly equivalent to the sequences $C_0, \ldots, C_k$ satisfying*

$$\begin{aligned} C_0 = \mu, \qquad C_k = 0, \qquad C_s \leq U_s \quad (0 \leq s \leq k), \\ 1 \geq d_1 \geq \cdots \geq d_k \geq 0, \qquad d_s = C_{s-1} - C_s, \end{aligned} \tag{13}$$

i.e. the sequence C_s is convex and dominated by U_s. The count probabilities can be recovered as

$$p_0 = 1 - \mu + C_1, \qquad p_r = C_{r-1} - 2C_r + C_{r+1}\ (1 \leq r < k), \qquad p_k = C_{k-1}. \tag{14}$$

Proof. For integer Z, we have

$$(Z - s + 1)_+ - (Z - s)_+ = \mathbf{1}_{\{Z \geq s\}}.$$

Consequently $d_s = \mathbb{P}(Z \geq s)$, so the differences satisfy the monotonicity inequalities, and the endpoints are simply $C_0 = \mathbb{E}Z = \mu$ and $C_k = 0$. Conversely, these conditions give $p_0 = 1 - d_1$, $p_r = d_r - d_{r+1}$ for $1 \leq r < k$, and $p_k = d_k$. The probabilities are nonnegative and sum to one. Summing their tails shows that their mean is $\sum_{s=1}^{k} d_s = \mu$ and their expected excess above s is $\sum_{t=s+1}^{k} d_t = C_s$, which give (14).

We next check the feasibility constraints. The identities

$$\mathbb{E}\min(s, Z) = \mu - C_s, \qquad \mathbb{E}\min(k - s, k - Z) = k - s - C_s$$

turn the inside and outside constraints into $C_s \leq \mu - Q_s$ and $C_s \leq k - s - G_{k-s}$. The count bounds in (1) ensure that extending the marginal lists by zeros introduces no additional restrictions, while constraints for groups larger than k follow from the mean equality. Thus, the original constraints are equivalent to $C_s \leq U_s$ for $0 \leq s \leq k$. □

Proof of Theorem 2. We first show that the largest convex sequence below U exists and satisfies (13). By Lemma A.1, there is at least one routing distribution that matches marginals, and we let $C^{(0)}$ be its expected-excess sequence. Consider all convex sequences below U, and let H_s be the supremum of their values at each s. This collection contains $C^{(0)}$ and is bounded above by U, so every H_s is finite. For any sequence C in the collection and $1 \leq s < k$,

$$2C_s \leq C_{s-1} + C_{s+1} \leq H_{s-1} + H_{s+1}.$$

Taking the supremum gives $2H_s \leq H_{s-1} + H_{s+1}$. Hence, H is convex and below U, and is the largest such sequence.

Because $C_s^{(0)} \leq H_s \leq U_s$, H also satisfies the endpoint conditions $H_0 = \mu$ and $H_k = 0$. Convexity makes the differences $H_{s-1} - H_s$ nonincreasing, and their extreme values satisfy

$$H_0 - H_1 \leq C_0^{(0)} - C_1^{(0)} \leq 1, \qquad H_{k-1} - H_k \geq C_{k-1}^{(0)} - C_k^{(0)} \geq 0. \tag{15}$$

Therefore, every consecutive difference lies in $[0, 1]$, and Lemma A.3 gives a feasible count distribution $p^\star$ with expected excess counts be exactly H.

It remains to show that this distribution uniquely maximizes the MGF for every $\theta > 0$. Using $C_s - C_{s+1} = \mathbb{P}(Z \geq s + 1)$, together with $C_0 = \mu$ and $C_k = 0$, we obtain

$$\begin{aligned}
\mathbb{E}e^{\theta Z} &= \sum_{r=0}^{k} p_r e^{\theta r} \\
&= 1 + (e^\theta - 1) \sum_{r=0}^{k} p_r \sum_{s=0}^{r-1} e^{\theta s} \\
&= 1 + (e^\theta - 1) \sum_{s=0}^{k-1} e^{\theta s} (C_s - C_{s+1}) \\
&= 1 + \mu(e^\theta - 1) + (e^\theta - 1)^2 \sum_{s=1}^{k-1} e^{\theta(s-1)} C_s.
\end{aligned} \tag{16}$$

The first two terms in (16) are already determined by the mean, and all remaining coefficients are strictly positive. Every feasible expected-excess sequence is bounded above by H, so $p^\star$ maximizes the MGF for every positive θ. If another feasible distribution attains the same maximum for any such θ, its expected excess counts must equal H at every internal threshold. Then (14) gives the same count distribution. □

Remark A.4. The distribution $p^\star$ is greatest in convex order among feasible count distributions (Shaked & Shanthikumar, 2007), while related work studies tail bounds for sums with fixed marginals and unknown dependence without the fixed top-k routing constraint (Puccetti & Rüschendorf, 2013).

Remark A.5 (A simpler MGF bound). Let $a_c < b_c$ be the lower and upper bounds on Z_c given in (1). By convexity we have for $\theta > 0$:

$$M_{\text{ext},c}(\theta) \le \frac{b_c - \mu_c}{b_c - a_c} e^{\theta a_c} + \frac{\mu_c - a_c}{b_c - a_c} e^{\theta b_c}.$$

Substituting the right hand side into (4) gives a conservative capacity using only the mean and these two bounds. This MGF bound is no tighter than $M_{\text{ext},c}(\theta)$.

Remark A.6 (Comparison of capacity bounds). We compare MarCap with the simpler MGF bound in Remark A.5 and an independent-expert approximation,

$$M_{\text{ind},c}(\theta) = \prod_{i \in \mathcal{C}_c} (1 - q_i + q_i e^{\theta}).$$

The independent-expert approximation uses the same marginals but ignores dependence between expert selections. We use all four middle-layer decode configurations from the existing risk-budget study. Within each configuration, all three calculations use the same calibration marginals and fixed LPT placement, with $\varepsilon = 0.01$ and the same per-card risk allocation and upward rounding in (4).

Table 5: Receive-buffer capacities for four decode configurations at $\varepsilon = 0.01$, in assignment rows per destination. Simpler uses Remark A.5; Indep. uses the independent-expert approximation.

Model/layer	EP	B	MarCap	Simpler	Indep.
Qwen L19	16	2048	1370	1385	1151
Qwen L19	32	2048	733	807	619
DeepSeek L14	8	4096	3599	3599	3317
DeepSeek L14	16	4096	1854	1863	1713

MarCap reduces the simpler capacity in three of the four configurations, by up to 9.17%, while the two capacities agree for DeepSeek L14/EP8. The independent-expert approximation gives smaller capacities in all four cases. Both MarCap and the simpler bound have zero observed overflow in the ten million held-out IID whole-route batches per configuration. At the independent-expert capacities, observed overflow ranges from 0.00404% to 0.02790%, also below the 1% budget in these four cases.

The largest convex sequence can be computed efficiently by the following stack construction: scan the points (s, U_s) from left to right. After adding each point, remove the middle of the last three retained points whenever the two successive slopes decrease, and repeat until the retained slopes are nondecreasing.

Proposition A.7 (Lower hull construction)**.** *Linear interpolation between the points retained by this construction gives the largest convex sequence below U. The scan and interpolation require $O(k)$ time and space. Including the sorting and prefix sums needed to form U gives $O(E \log E + k)$ preparation time per card.*

Proof. For three successive retained indices $a < b < c$, write $\sigma(a, b) = \frac{U_b - U_a}{b - a}$. We have

$$U_b - \left(\frac{c - b}{c - a} U_a + \frac{b - a}{c - a} U_c \right) = \frac{(b - a)(c - b)}{c - a} \big(\sigma(a, b) - \sigma(b, c) \big). \tag{17}$$

Thus, when the slopes decrease, the middle point must be above the chord joining the other two. Replacing the two segments by this chord lowers the interpolation throughout that interval, so it remains below every previously processed upper-bound point. Adding a new point extends the interpolation only to the next integer index, and induction shows that the final interpolation lies below all the bounds U_s. The slopes are nondecreasing by construction and guarantees convexity.

Now take any convex sequence C below U. Between two consecutive retained indices $a < c$, convexity gives

$$C_s \le \frac{c - s}{c - a} C_a + \frac{s - a}{c - a} C_c \le \frac{c - s}{c - a} U_a + \frac{s - a}{c - a} U_c, \qquad a \le s \le c.$$

The last expression is the stack's interpolation, so it is at least as large as every other convex sequence below U. Because each point is added once and removed at most once, with interpolation visiting each integer index once, both operations require only $O(k)$ time and space. Sorting the two marginal lists and computing their prefix sums gives the stated total preparation time. □

Therefore, MarCap can be summarized by the pseudocode below.

Algorithm 1: MarCap
Input: expert marginals q, placement, k, batch size B, risk budget ε.
1. For each card, sort its inside and outside marginals and compute U_s.
2. Scan (s, U_s) with a stack; remove a middle point when slopes decrease.
3. Interpolate the retained vertices and recover $p_c^\star$ by (14).
4. Form $M_{\mathrm{ext},c}(\theta) = \sum_r p^\star_{c,r} e^{\theta r}$.
5. Compute the scalar infimum for each card using Proposition A.9 and apply (4).
Output: common receive-buffer capacity W_{mar}.

Example A.8. Figure 6 uses $k = 4$ and $q = \left(1, \frac{1}{3}, \frac{1}{3}, \frac{1}{3}, 1, \frac{1}{3}, \frac{1}{3}, \frac{1}{3}\right)$, with the first four experts on the target card. The bounds are $U = \left(2, 1, \frac{2}{3}, 0, 0\right)$. Although $U_2 = \frac{2}{3}$, convexity requires $C_2 \le \frac{C_1 + C_3}{2} \le \frac{1}{2}$. The sequence $C^\star = \left(2, 1, \frac{1}{2}, 0, 0\right)$ is convex and satisfies all the bounds, attaining the largest possible value at each coordinate. Formula (14) gives $p_1^\star = p_3^\star = \frac{1}{2}$ and all other probabilities zero.

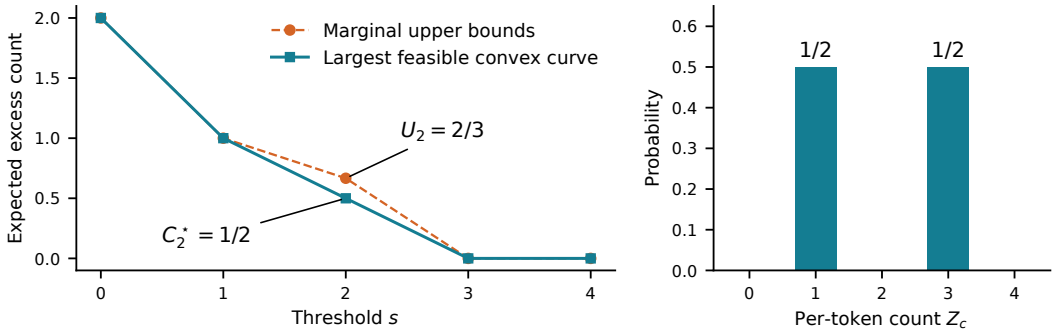


Figure 6: **The lower hull computes the same MGF optimum as the LP.** In the top-4 example above, each card holds one always-selected expert and three experts selected with probability $\frac{1}{3}$. Left: the marginal upper bounds and the largest feasible convex sequence of expected excess counts. Convexity limits the value at threshold 2 to $\frac{1}{2}$. Right: the recovered per-token count probabilities.

A.3 Risk-bounded capacity and scalar minimization

Once the MGF-maximizing distribution is known, computing W_{mar} in (4) requires optimizing $\theta > 0$ for each card. For a fixed card, write $\psi(\theta) = \log M_{\mathrm{ext},c}(\theta) = \log \sum_r p_r^\star e^{\theta r}$ and $a = \log\left(\frac{\mathrm{EP}}{\varepsilon}\right) > 0$, where $\varepsilon \in (0, 1)$. Thus, we seek the infimum of $F(\theta) = \frac{B\psi(\theta) + a}{\theta}$ over $\theta > 0$, where B is the batch size.

Proposition A.9 (Scalar minimization)**.** *Let r_+ be the largest count with $p^\star_{r_+} > 0$, and write $p_+ = p^\star_{r_+}$. The function F has a finite minimizer on $\theta > 0$ if and only if*

$$0 < \frac{a}{B} < -\log p_+. \tag{18}$$

When it exists, this minimizer θ^* is unique and satisfies

$$\theta^* \psi'(\theta^*) - \psi(\theta^*) = \frac{a}{B}, \qquad F(\theta^*) = B\psi'(\theta^*). \tag{19}$$

Otherwise, F decreases to the unattained infimum Br_+ as $\theta \to \infty$.

Proof. Define the probabilities $p_{\theta,r} = \frac{p_r^\star e^{\theta r}}{M_{\mathrm{ext},c}(\theta)}$, and let $\mathbb{E}_\theta$ and Var_θ denote the expectation and the variance under this distribution. Differentiating the finite sum gives

$$\psi'(\theta) = \mathbb{E}_\theta Z_c, \qquad \psi''(\theta) = \mathrm{Var}_\theta(Z_c), \qquad F'(\theta) = \frac{B(\theta\psi'(\theta) - \psi(\theta)) - a}{\theta^2}. \tag{20}$$

Suppose first that $p^\star$ is not concentrated at one particular count. Then its variance remains positive after reweighting, so $h(\theta) = \theta\psi'(\theta) - \psi(\theta)$ satisfies

$$h(0) = 0, \qquad h'(\theta) = \theta \operatorname{Var}_\theta(Z_c) > 0.$$

To determine its upper limit, factor out the largest exponential:

$$\psi(\theta) = \theta r_+ + \log\left(p_+ + \sum_{r<r_+} p_r^\star e^{-\theta(r_+-r)}\right). \tag{21}$$

The sum over $r < r_+$ and its derivative decay exponentially. Consequently $\theta(\psi'(\theta) - r_+) \to 0$, and

$$\lim_{\theta\to\infty} h(\theta) = -\log p_+, \qquad \lim_{\theta\to\infty} F(\theta) = Br_+.$$

Furthermore, $F(\theta) \to \infty$ as $\theta \to 0$ because $\psi(0) = 0$ and $a > 0$. Thus, F' changes from negative to positive at exactly one point if (18) holds. We then have the unique minimum and the first identity in (19), and substituting it into F gives the second identity. Moreover, if $\frac{a}{B} \geq -\log p_+$, we get $F' < 0$ for every finite θ.

Finally, if $p_+ = 1$, then $F(\theta) = Br_+ + \frac{a}{\theta}$, which also decreases to Br_+ as $\theta \to \infty$. □

Corollary A.10 (Capacity guarantee)**.** *For B IID tokens whose routing distribution belongs to $\mathcal{P}(q)$, the common receive-buffer capacity $W_{\mathrm{mar}}(\varepsilon)$ in (4) satisfies*

$$\mathbb{P}\Big(\max_c L_c > W_{\mathrm{mar}}(\varepsilon)\Big) \leq \varepsilon.$$

Proof. For a fixed card c, the MGF of each token count is bounded by $M_{\mathrm{ext},c}(\theta)$. Token independence and Markov's inequality give

$$\mathbb{P}(L_c > W) \leq \exp\big(-\theta W + B\psi(\theta)\big), \qquad \theta > 0.$$

If the scalar infimum for this card is attained at θ^*, then $W_{\mathrm{mar}} \geq F(\theta^*)$ by (4), so such probability is at most $e^{-a} = \frac{\varepsilon}{\mathrm{EP}}$.

If the infimum is not attained, Proposition A.9 gives its value as Br_+. For any feasible count distribution, Theorem 2 and its proof have $0 \leq C_{r_+} \leq C^\star_{r_+} = 0$, so $\mathbb{E}[(Z_c - r_+)_+] = 0$, which means $Z_c \leq r_+$ almost surely. Because $W_{\mathrm{mar}} \geq Br_+$, overflow on this card is impossible.

Thus, we can bound each card's overflow probability by $\frac{\varepsilon}{\mathrm{EP}}$ under every routing distribution in $\mathcal{P}(q)$, then applying the union bound yields the result. Note that this step uses only the individual bounds and requires no independence between cards. □

Remark A.11 (Reuse and numerical evaluation). The maximizing distributions depend only on the marginals and placement. Thus, Steps 1–3 of Algorithm 1 can be reused when only B or ε changes. With these distributions stored, I_θ scalar iterations over all cards cost $O(I_\theta \cdot \mathrm{EP} \cdot k)$. We evaluate the finite exponential sums using log-sum-exp and solve (19) when a finite minimum exists. A risk guarantee covering several layers or invocations requires dividing the risk budget across them as well.

Remark A.12 (Uniform balls into bins). Suppose $k = 1$, $\mathrm{EP} = E$ and $q_i = \frac{1}{E}$, with one expert on each card. For B IID tokens, routing is identical to placing B balls independently and uniformly into E bins. For fixed $\varepsilon \in (0, 1)$ and $E \to \infty$, the capacity in (4) recovers the classical results (Gonnet, 1981; Raab & Steger, 1998):

$$W_{\mathrm{mar}}(\varepsilon) \sim \begin{cases} \dfrac{\log E}{\log\log E}, & B = E, \\ \dfrac{B}{E}, & \dfrac{B}{E\log E} \to \infty. \end{cases}$$

To see this, each card has a Bernoulli count with parameter $\frac{1}{E}$, so $\psi(\theta) = \log\left(1 + \frac{e^\theta - 1}{E}\right)$. Write $w = \inf_{\theta>0} F(\theta)$, so $W_{\mathrm{mar}} = \lceil w \rceil$. For sufficiently large E in either case, the minimum is attained and (19) gives $w = B\psi'(\theta^*)$. Substituting the Bernoulli MGF into the same optimality equation yields

$$w \log\frac{Ew}{B} + (B - w)\log\frac{B - w}{B(1 - 1/E)} = \log\frac{E}{\varepsilon}.$$

When $B = E$, evaluating F at $\theta = 1$ gives $w \le F(1) = O(\log E)$. Therefore, the last identity becomes $w \log w - w + 1 + o(1) = \log(E/\varepsilon)$, which gives the first limit. When $\frac{B}{E \log E} \to \infty$, $\theta^* \to 0$, and expanding ψ at zero in (19) gives

$$W_{\mathrm{mar}}(\varepsilon) = \frac{B}{E} + \sqrt{\frac{2B}{E}\left(1 - \frac{1}{E}\right)\log\frac{E}{\varepsilon}} + O\left(\log\frac{E}{\varepsilon}\right).$$

The correction is of smaller order than B/E when $B \gg E \log E$, giving the second limit.

B Numerical agreement and CPU planning cost

B.1 Full LP versus reduced LP

We test the reduction in Theorem 1 on small systems where the full LP can be solved by enumerating all size-k expert subsets. We generate 1000 random cases with $2 \le E \le 9$, $1 \le k \le E$ and $1 \le \mathrm{EP} \le \min(E, 4)$. We construct feasible marginals from dense or sparse mixtures of the subsets and assign experts randomly to cards. Three additional fixed cases cover deterministic routing, uniform subset selection and a card with a positive minimum count.

These cases give 2243 per-card problems. For each, we compare the full LP and the reduced count LP using five general linear objectives, giving 11215 comparisons. We also compare MGF objectives at $\theta \in \{10^{-5}, 10^{-3}, 0.02, 0.1, 0.5, 1, 3, 10, 30\}$, giving 20187 card–parameter comparisons. MGF values are multiplied by $e^{-\theta k}$ before comparison to avoid very large exponentials. Table 6 reports the numerical discrepancies.

Table 6: Full LP versus reduced LP: absolute differences in the optimal objective values. MGF differences use the scaling described above.

Objective	Comparisons	Maximum difference
General linear objectives	11215	1.73×10^{-14}
Scaled MGF objectives	20187	$\le 9.71 \times 10^{-10}$

B.2 Numerical agreement of the three methods

We next compare the numerical outputs of the three methods in Table 1: the reduced LP at every θ, one LP per card model followed by reuse of its count distribution, and direct hull construction. All three methods use the same calibration marginals and placements for Qwen prefill layer 19 and DeepSeek prefill layer 14, with the same batch sizes and EP settings as the routing study and a risk budget of 0.01. The one-LP and hull count distributions are compared componentwise, and their scaled MGFs are compared at the same nine parameter values used above. For these MGF comparisons, we replace k in the scaling factor $e^{-\theta k}$ by the largest count whose computed probability exceeds 10^{-13}.

For capacities, we compare all three method pairs over 7140 per-card and batch-size combinations for Qwen and 2268 for DeepSeek. Taking the maximum over cards and rounding up gives 112 common integer capacities for Qwen and 108 for DeepSeek. Table 7 shows that these integer capacities agree exactly for every method pair, while the differences before rounding remain small.

Table 7: Numerical agreement of the three methods. The first five rows give maximum absolute differences. Continuous capacities are compared per card before rounding, in assignment rows. The last row gives mismatches / common integer capacities for each method pair.

Quantity	Comparison	Qwen L19	DeepSeek L14
Count probabilities	One LP vs. Hull	1.11×10^{-15}	6.66×10^{-16}
Scaled MGF	One LP vs. Hull	9.99×10^{-16}	8.88×10^{-16}
Continuous capacity	LP at each θ vs. One LP	1.82×10^{-8}	3.81×10^{-7}
	LP at each θ vs. Hull	2.08×10^{-8}	3.81×10^{-7}
	One LP vs. Hull	1.56×10^{-8}	3.28×10^{-7}
Integer mismatches	Each method pair	0/112	0/108

The timing experiment runs on a 32-logical-processor AMD64 host using Python, NumPy and SciPy. Each timing includes distribution preparation and the scalar capacity calculation, but excludes routing analysis and placement. Following a warmup, we randomize method order in each of five paired repetitions. Table 1 reports median time and median paired speedup. Relative to an LP at every θ, the full hull method is 90.1–$98.3\times$ faster for Qwen and 106.6–$113.7\times$ for DeepSeek across repetitions. Relative to one LP per card model, the hull provides an additional median paired speedup of $8.50\times$ and $5.46\times$, respectively.

C A BATCH-LOAD BOUND FOR INDEPENDENT BLOCKS

The theory in Appendix A could perform poorly in prefill when adjacent tokens have correlated routes and traffic mixing is insufficient. The same issue could arise in decode as well when more aggressive speculative decoding is deployed, where each request contributes several tokens to a processed batch. Therefore, we extend the risk guarantee by assuming independence at the request level instead of at the individual token level. In this section, we model each request's contribution as one block, allow arbitrary dependence within the block, and require an upper bound on its length. Thus, for prefill this requires a maximum prompt length or an enforced per-request chunk limit, while for speculative decoding it requires a maximum number of tokens processed per request in a batch.

We partition a fixed batch into blocks $I_1, \dots, I_J$ and let $X_{j,c} = \sum_{t \in I_j} Z_c^{(t)}$ be block j's contribution to card c. If the blocks are independent and $g_{j,c}(\theta)$ is an upper bound on $\log \mathbb{E}e^{\theta X_{j,c}}$, then

$$\log \mathbb{E}e^{\theta L_c} \le \sum_j g_{j,c}(\theta). \tag{22}$$

The following result obtains a block bound from the same single-token MGF used in the main text.

Proposition 3 (Independent blocks with bounded length)**.** *Suppose the blocks are independent, $b_j = |I_j| \le \ell$, and every token's routing distribution belongs to $\mathcal{P}(q)$. We have for $\theta > 0$,*

$$\log \mathbb{E}e^{\theta L_c} \le \sum_j \log M_{\mathrm{ext},c}(b_j\theta) \le \frac{B}{\ell} \log M_{\mathrm{ext},c}(\ell\theta). \tag{23}$$

Therefore, a conservative common capacity is

$$W_{\mathrm{block}} = \left\lceil \max_c \inf_{\nu>0} \frac{B \log M_{\mathrm{ext},c}(\nu) + \ell \log(\mathrm{EP}/\varepsilon)}{\nu} \right\rceil. \tag{24}$$

Proof. Applying the generalized Hölder inequality within block I_j gives

$$\mathbb{E} \prod_{t \in I_j} e^{\theta Z_c^{(t)}} \le \prod_{t \in I_j} \left(\mathbb{E}e^{b_j \theta Z_c^{(t)}} \right)^{1/b_j} \le M_{\mathrm{ext},c}(b_j\theta). \tag{25}$$

The first inequality in (23) comes from independence between blocks. Because $\log M_{\mathrm{ext},c}$ is convex and zero at the origin, we have

$$\log M_{\mathrm{ext},c}(b_j\theta) \le \frac{b_j}{\ell} \log M_{\mathrm{ext},c}(\ell\theta). \tag{26}$$

Summing over all blocks gives the second inequality in (23). Finally, substituting $\nu = \ell\theta$ into the Chernoff calculation gives (24). □

The original formula is recovered when $\ell = 1$. A larger upper bound ℓ on block length increases the risk term in (24), making the capacity bound more conservative. At a fixed batch size, mixing more independent requests can reduce the maximum number of tokens contributed by any one request and hence the capacity overhead.

D TOKEN-DEDUPLICATED RECEIVE BUFFERS

The main text measures receive-buffer capacity in token–expert assignment rows. If a token selects several experts on one card, sending a separate copy for each assignment duplicates its input on that card. A communication operator can instead send the token once per destination card and reuse it for the local experts. This is particularly useful at small EP: a token selects k experts but reaches at most $\min(k, \mathrm{EP})$ cards, so repeated destinations are unavoidable when $\mathrm{EP} < k$. We want to explore in this part whether we can still use the expert marginals to determine a smaller capacity for this receive buffer or whether we need additional routing information.

Routing telemetry can provide information about expert co-selection beyond individual marginals and help improve expert placement. MoETuner profiles expert transitions between neighboring layers for placement, while Gimbal collects source-to-expert traffic for communication-aware placement and scheduling (Go & Mahajan, 2025; Sun et al., 2026). For deduplicated receive buffers, the useful additional observation is whether a token reaches each card, which depends on which experts on the same card are selected together. We first establish what expert marginals alone can guarantee, then evaluate capacities chosen using this additional information.

D.1 CAPACITY FROM EXPERT MARGINALS

Let $H_c = \mathbf{1}_{\{Z_c>0\}}$ indicate whether a token reaches card c, and write $h_c = \mathbb{P}(Z_c > 0)$. The deduplicated batch load is

$$L_c^{\mathrm{hit}} = \sum_{t=1}^{B} H_c^{(t)}, \qquad W_{\max}^{\mathrm{hit}} = B. \tag{27}$$

Thus, full reservation now allocates B token slots per card rather than $B\min(k, m)$ assignment rows. The required single-token MGF depends only on h_c. However, expert marginals do not determine h_c, and the count distribution that maximizes $M_{\mathrm{ext},c}$ does not necessarily maximize the probability of reaching the card.

Proposition 4 (Deduplicated reception from expert marginals)**.** *For feasible expert marginals q and a fixed placement, the maximum probability of reaching card c and the corresponding MGF are, for $\theta > 0$,*

$$\begin{aligned} h_c^{\max} &= \max_{P\in\mathcal{P}(q)} \mathbb{P}_P(Z_c > 0) = \min(1, \mu_c), \\ \max_{P\in\mathcal{P}(q)} \mathbb{E}_P e^{\theta H_c} &= 1 + (e^\theta - 1)h_c^{\max}. \end{aligned} \tag{28}$$

Proof. Because $H_c \le Z_c$ and $H_c \le 1$, we have $\mathbb{E}H_c \le \min(1, \mu_c)$. It remains to show that a routing distribution with the given marginals can attain this upper bound. When $\mu_c \le 1$, it is enough to select at most one expert on card c for every token, and when $\mu_c \ge 1$, it is enough to select at least one expert on the card for every token.

Consider the polytope defined by $x \in [0,1]^E$ and $\sum_i x_i = k$, with the additional constraint $\sum_{i\in\mathcal{C}_c} x_i \le 1$ in the first case or $\sum_{i\in\mathcal{C}_c} x_i \ge 1$ in the second. The vector q belongs to the corresponding polytope. By the same argument as in Lemma A.1, every vertex is a vector with only zeros and ones and exactly k ones. When the added constraint holds with equality, the sums inside and outside $\mathcal{C}_c$ are integers, so the two fractional coordinates in that argument can be chosen from the same group.

Express q as a convex combination of these vertices and choose the corresponding expert sets with those coefficients as probabilities. This preserves all expert marginals and selects exactly k experts.

For $\mu_c \le 1$, the group constraint gives $H_c = Z_c$ and $h_c = \mu_c$. For $\mu_c \ge 1$, it gives $H_c = 1$ almost surely. Finally, $\mathbb{E}e^{\theta H_c} = 1 + (e^\theta - 1)h_c$, which is increasing in h_c for $\theta > 0$, gives the MGF formula. □

Under the IID token assumption, L_c^{hit} is binomial with parameter h_c, and replacing h_c by $h_c^{\max}$ bounds its upper tail. If $\mu_c \ge 1$, this gives full reservation on that card. In particular, $\mathrm{EP} \le k$ forces at least one $\mu_c \ge 1$ because $\sum_c \mu_c = k$, so the common capacity based only on expert marginals is B. In these cases, deduplication can still reduce the receive buffer relative to dispatch without token deduplication, but further reduction below B requires additional information about routing.

D.2 Experimental setup and metrics

We use the same two-model routing records, 25% calibration request split, model layers, batch sizes and fixed LPT placements as the main routing study. After constructing placement from the calibration expert marginals, we count whether each calibration token reaches each card. From these counts we estimate h_c and compute one-sided binomial upper confidence bounds $\bar{h}_c$. We then choose

$$W_{\mathrm{hit}} = \max_c Q_{1-\varepsilon/\mathrm{EP}}\big(\mathrm{Binomial}(B, \bar{h}_c)\big), \tag{29}$$

where Q is the smallest integer whose cumulative probability reaches the specified level. Conditional on the upper bounds covering the actual h_c, this capacity gives overflow probability at most ε for IID tokens. We use a nominal calibration error budget of 10^{-6} per model, allocated across layers, stages and destination cards. The binomial confidence calculation requires independent calibration observations and a placement fixed independently of them. Our placement is derived from calibration data, and recorded token routes may be dependent, so we evaluate the observed overflow on held-out traffic rather than claim this calibration guarantee for the recorded workloads.

Capacities remain fixed during evaluation and the risk budget is 0.01. We retain the three settings used in the main text: contiguous prefill, shuffled prefill, and decode batches sampled independently from complete held-out token routes. Decode uses 10000 batches per configuration. Deduplication does not change the recorded expert selections or placement; it changes whether we count every selected expert or each destination card once. This gives 1008 Qwen and 972 DeepSeek tested configurations.

As in the main text, we normalize the capacity factor by the expected card load under uniform random expert choices. With E/EP experts per card, the probability of reaching a card under uniform selection of k distinct experts is

$$h_{\mathrm{unif}} = 1 - \frac{\binom{E - E/\mathrm{EP}}{k}}{\binom{E}{k}}, \qquad \mathrm{CF}_{\mathrm{hit}}(W) = \frac{W}{B h_{\mathrm{unif}}}. \tag{30}$$

Let $W_{\mathrm{req}}^{\mathrm{hit}}(\varepsilon)$ be the smallest capacity whose observed deduplicated overflow frequency is at most ε. The compression rate, CF gap and observed overflow probability are

$$\begin{aligned} \eta_{\mathrm{hit}} &= 1 - \frac{W_{\mathrm{hit}}}{B}, \qquad \Delta\mathrm{CF}_{\mathrm{hit}} = \frac{W_{\mathrm{req}}^{\mathrm{hit}} - W_{\mathrm{hit}}}{B h_{\mathrm{unif}}}, \\ \widehat{o}_{\mathrm{hit}} &= \frac{1}{n}\sum_{j=1}^{n} \mathbf{1}_{\{\max_c L_c^{\mathrm{hit},(j)} > W_{\mathrm{hit}}\}}. \end{aligned} \tag{31}$$

Thus, $\Delta\mathrm{CF}_{\mathrm{hit}} \le 0$ retains the same interpretation: the observed overflow frequency does not exceed the risk budget.

For the receive-buffer comparison in MiB, we count BF16 token vectors of hidden width 2048 for both models. Therefore, each receive slot occupies 4096 bytes, giving a capacity saving of $(B - W_{\mathrm{hit}})/256$ MiB against deduplicated full reservation. This estimate excludes GEMM buffers, routing indices and fixed overhead, so it is not a measurement of total operator scratch workspace. The percentages in (31) likewise measure additional reduction below B, not the reduction already obtained by removing repeated copies.

D.3 RESULTS

Figure 7 shows the compression rates for the middle captured layers at EP4, EP16 and EP32. At EP4 and EP32, the planned compression rates at the largest prefill batch size are 7.12% and 75.65% for Qwen, and 14.27% and 80.85% for DeepSeek, respectively. These savings below B show the value of measuring whether a token reaches a card: expert marginals alone require full reservation at EP4. For the same largest-prefill EP4 configurations, the planned capacities without token deduplication are 67742 and 794631 assignment rows, whereas the deduplicated capacities are 30435 and 449458 token slots for Qwen and DeepSeek. These correspond to receive-buffer capacity reductions of 55.07% and 43.44% relative to the plans without token deduplication.

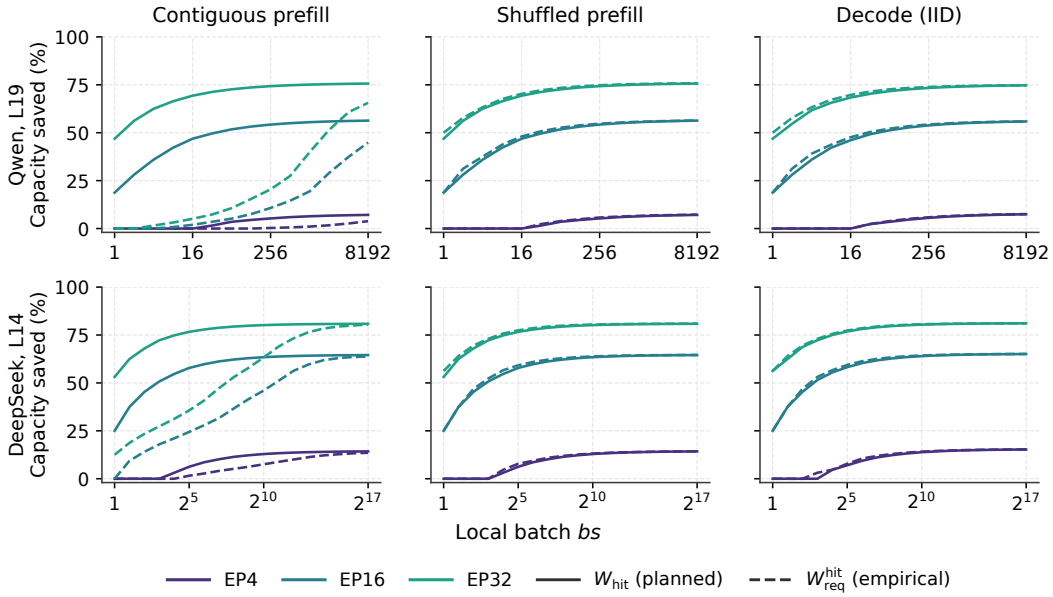


Figure 7: **Receive-buffer compression rates with deduplication.** Middle captured layers, EP4, EP16 and EP32. Solid curves use W_{hit} and dashed curves use $W^{\text{hit}}_{\text{req}}$, both normalized by the deduplicated full reservation B. Colors identify EP as in Figure 8. The plotted savings concern the receive buffer, not GEMM buffers.

Figure 8 complements the compression curves with CF gaps, using the same scatter-plot encodings as Figure 2. Contiguous prefill frequently requires more capacity than planned: 94/112 Qwen L19 configurations and 94/108 DeepSeek L14 configurations exceed the 1% budget. Shuffled prefill and decode generally have nonpositive gaps in these middle layers. Nevertheless, two DeepSeek L14 shuffled-prefill configurations exceed the budget; its maximum CF gap is only 0.00123, while observed overflow reaches 8.75%. As in the main text, a small positive CF gap should not be interpreted as a small risk-budget violation.

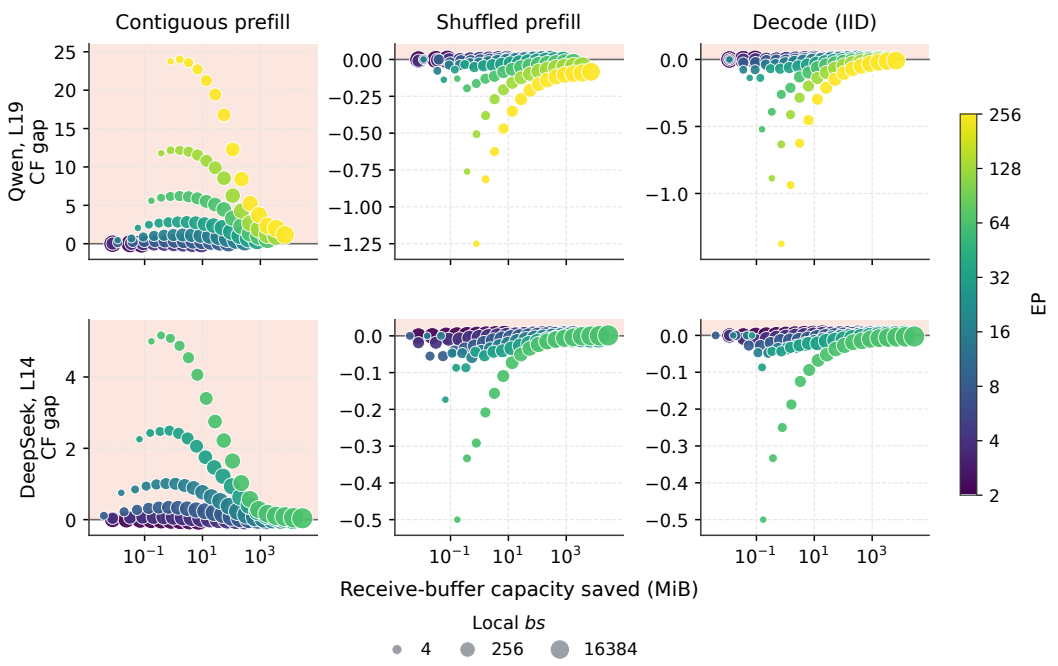


Figure 8: **CF gaps and receive-buffer capacity savings with deduplication.** Each point is a middle-layer batch-size/EP configuration. Horizontal position is the calculated saving for BF16 token vectors of width 2048, excluding GEMM buffers and metadata. Color denotes EP and marker size denotes local bs. Positive CF gaps are shaded. All 565 positive-saving configurations are shown; 95 zero-saving configurations are omitted from the logarithmic horizontal axis.

Table 8 gives the CF and CF gap for selected configurations at two batch sizes. The common prefill capacity applies to both contiguous and shuffled versions of the same calibration/evaluation split, while their empirical gaps differ. At EP32, increasing local batch size from 128 to 8192 reduces the contiguous-prefill gap from 2.56195 to 0.44301 for Qwen L19 and from 1.75045 to 0.23230 for DeepSeek L14. The gaps remain positive even though larger batches mitigate the shortfall.

Table 8: Capacity factors and CF gaps for selected deduplicated configurations, using (30)–(31). Prefill CF is shared by contiguous and shuffled prefill.

Model/layer	EP	Local bs	Prefill CF	Contig. CF gap	Shuffled CF gap	Decode CF	Decode CF gap
Qwen L19	4	128	1.05702	+0.04972	−0.00432	1.05486	−0.00432
Qwen L19	4	8192	1.02794	+0.03634	−0.00138	1.02430	−0.00108
Qwen L19	32	128	1.16110	+2.56195	−0.02580	1.20303	−0.03010
Qwen L19	32	8192	1.07246	+0.44301	−0.00764	1.11269	−0.00275
DeepSeek L14	4	128	1.07427	+0.07240	−0.00934	1.06259	−0.00934
DeepSeek L14	4	8192	1.02990	+0.03124	−0.00124	1.01818	−0.00146
DeepSeek L14	32	128	1.17827	+1.75045	−0.03390	1.16335	−0.02169
DeepSeek L14	32	8192	1.07439	+0.23230	−0.00737	1.06081	−0.00415

The middle-layer decode configurations shown in the scatter remain within the budget, but this does not hold across all captured layers. Across all tested configurations, 10/336 Qwen decode configurations and 15/324 DeepSeek decode configurations exceed 1%, with maximum observed overflow of 94.55% and 99.80%. Shuffled-prefill maxima over all layers are 0.402% and 8.75%, respectively. Additional routing information can reduce receive-buffer reservation, but the resulting capacities remain sensitive to inter-token dependence and calibration-to-evaluation marginal drift.

E REPLICATED EXPERTS AND EXPERT-PARALLEL LOAD BALANCING

Expert-parallel load balancing can replicate heavily used experts and distribute their copies across cards (DeepSeek-AI, 2026b). This can spread their assignments without changing which logical experts a token selects. However, the number and locations of the copies do not by themselves determine each card's load because dispatch must also choose which copy executes each assignment. Thus, we want to investigate how to modify the capacity calculation to account for this choice, and how do different replication configurations affect the planned and empirical capacities. In this section, we extend the count model to physical routing marginals, then use a controlled routing study to compare four replica counts under two distinct dispatch rules. The key convention remains the same: each token still selects exactly k logical experts, with one physical execution per selected expert.

E.1 COUNT FEASIBILITY WITH PHYSICAL ROUTING MARGINALS

Let Γ_i be the cards holding copies of logical expert i, and let Y_{ic} indicate whether the token is processed by a copy of expert i on card c. We make $X_i \in \{0,1\}$ indicates logical selection, which gives

$$Y_{ic} = 0 \ (c \notin \Gamma_i), \qquad \sum_c Y_{ic} = X_i, \qquad \sum_i X_i = k. \tag{32}$$

Write $a_{ic} = \mathbb{P}(Y_{ic} = 1)$. The logical marginals should satisfy $\sum_c a_{ic} = q_i \le 1$ and $\sum_i q_i = k$, while the load on a target card is $Z_c = \sum_i Y_{ic}$. For this specific card, we define $q_i^{\text{in}} = a_{ic}$ and $q_i^{\text{out}} = q_i - a_{ic}$ as the probabilities that expert i executes on or off the card. Their inside sum is $\mu_c = \sum_i q_i^{\text{in}}$. As in Appendix A, we let Q_s and G_s be the sums of the s largest inside and outside probabilities. It is worth noting that unlike the nonreplicated case, both probabilities for a logical expert can be positive. Therefore, we need to preserve the restriction that a token never selects both its inside and outside execution.

Proposition 5 (Count feasibility with replicated experts)**.** *Among routing distributions with the specified physical marginals, count probabilities $p_r = \mathbb{P}(Z_c = r)$ are realizable exactly when they form a probability vector of mean μ_c and*

$$Q_s \le \sum_r \min(s, r) p_r, \qquad G_s \le \sum_r \min(s, k-r) p_r \quad (1 \le s \le E). \tag{33}$$

Here the dispatch rule is free to choose any available copy, subject to one execution per selected logical expert. The MGF-maximizing distribution is recovered from the lower convex hull of $U_s = \min\{\mu_c - Q_s,\ k - s - G_{k-s}\}$ for $0 \le s \le k$ using Appendix A.2.

Proof. Necessity follows by counting selected logical experts executed on and off the target card, as in Theorem 1. To prove sufficiency, we first construct the inside and outside selections and then remove any overlap between them.

Suppose $(q^{\text{in}}, q^{\text{out}}, p)$ are rational, with common denominator d. We create d token columns, of which dp_r require r inside and $k-r$ outside selections. The integer flow construction produces two matrices whose entries are only zeros and ones, such that rows index logical experts, columns index tokens, and a one denotes an inside or outside selection. Their row sums are dq_i^{in} and dq_i^{out} respectively, and each combined column contains k selections. The two matrices may initially select the same expert in the same column. We pick a pair with the smallest number of such overlaps, and let $m_{ij} \in \{0,1,2\}$ be the sum of their entries.

If $m_{ij} = 2$, the row sum $dq_i \le d$ implies that another column j' has $m_{ij'} = 0$. Because both combined columns sum to k, some row i' must satisfy $m_{i'j'} > m_{i'j}$. Then we select an inside or outside matrix with ones at (i', j') and (i, j) and zeros at (i', j) and (i, j'). Moving these two ones from (i, j) and (i', j') to (i, j') and (i', j) preserves the row and column sums and removes the overlap at (i, j) without creating one at (i, j'). If this creates an overlap at (i', j), then previously $m_{i'j} = 1$ and $m_{i'j'} = 2$, so the same move also removes the overlap at (i', j'). Therefore, the total number of overlaps decreases which contradicts the minimality. Hence, the selections can be made disjoint and sampling a column uniformly shall realize the required count law and marginals.

For arbitrary real inputs, the prefix conditions can equivalently be written as linear inequalities for all expert subsets. They define a polytope with rational coefficients, so feasible real inputs can then

be approximated using feasible rational inputs. Then the corresponding routing distributions would have a convergent subsequence in the finite probability simplex, and the limit realizes the desired inputs. For an expert selected outside the target card, simply choose another destination c' with probability $a_{ic'}/q_i^{\text{out}}$ can recover its remaining physical marginals, and such choice is only needed when $q_i^{\text{out}} > 0$. Therefore, the feasible count probabilities have the same prefix constraints as in Appendix A.2, and its expected-excess argument gives the stated MGF maximizer. □

The proposition allows every dispatch choice consistent with the physical marginals, but a fixed dispatch rule could have more restrictions. Still, the maximum over the larger set remains an upper bound for that rule. If only the replica locations are known, we instead have the looser bound $Z_c \le |S \cap \{i : c \in \Gamma_i\}|$. Thus, additional copies alone doesn't necessarily establish a smaller receive-buffer capacity.

E.2 Source-dependent dispatch

Suppose each source rank s uses a deterministic rule $\sigma_s(i) \in \Gamma_i$ to select the destination of logical expert i, and we define $\mathcal{C}_{c,s} = \{i : \sigma_s(i) = c\}$. For a fixed source, every logical expert has exactly one destination. Therefore, Theorem 1 applies with $\mathcal{C}_c$ replaced by $\mathcal{C}_{c,s}$ and with that source's expert marginals. Let $M_{\text{ext},c,s}$ be the resulting worst-case single-token MGF. With independent tokens and fixed source batch sizes B_s, the source contributions satisfy

$$\begin{aligned} \log \mathbb{E}e^{\theta L_c} &\le \sum_s B_s \log M_{\text{ext},c,s}(\theta), \\ W_{\max} &= \max_c \sum_s B_s \min(k, |\mathcal{C}_{c,s}|). \end{aligned} \tag{34}$$

The capacity calculation then uses this sum in place of $B \log M_{\text{ext},c}(\theta)$. Full reservation also depends on the dispatch rule, because it determines which logical experts each source can send to a destination.

E.3 Experimental setup and metrics

We use the DeepSeek-V2-Lite-Chat EP8 routing study with $E = 64$, $k = 6$, and layers 3, 14 and 26. The four configurations add 0, 16, 32 or 64 replicas, giving 64, 80, 96 or 128 physical expert copies. A load-based plan repeatedly adds a copy to the expert with the largest estimated load per copy. Constrained LPT then places equal numbers of expert copies on each card and keeps copies of the same logical expert on different cards. Replication and placement are computed separately for each layer, stage and replica count, and remain fixed during evaluation.

At each fixed replication configuration, two dispatch rules share the physical placement and the ordered copies of every expert. *Adjacent* dispatch assigns consecutive groups of source ranks to a copy, while *Interleaved* dispatch assigns source ranks cyclically across copies. For example, with two copies, *Adjacent* sends sources 0–3 to the first and 4–7 to the second, whereas *Interleaved* separates even and odd source ranks. More generally, if expert i has n_i copies, *Adjacent* uses consecutive groups of $\lceil 8/n_i \rceil$ sources, while *Interleaved* selects the copy indexed by the source rank modulo n_i.

Calibration uses a separate 25% request split from the main routing study. Local batch sizes are powers of two from 16 to 131072, with $B = 8bs$. Prefill evaluates up to 2000 complete, nonoverlapping batches where each global batch is divided into eight consecutive source groups of bs tokens. Shuffled prefill changes token order, while decode uses 2000 batches sampled with replacement from complete held-out token routes. At a fixed mode and batch size, all replica counts and both dispatch rules use the same logical routing samples. There are 504 configurations per rule and 1008 in total.

The planner uses the same calibration expert marginals for all sources, while their destination sets differ through σ_s. With $\varepsilon = 0.01$, it selects the common integer capacity W by applying the Chernoff bound to $L_c \ge W + 1$.

Again let W_{req} denote the smallest capacity whose observed overflow frequency is at most ε. Replication preserves the total Bk token–expert assignments, so the CF normalization remains the same as in the main text:

$$\text{CF}(W) = \frac{W}{Bk/\text{EP}}, \qquad \Delta\text{CF} = \frac{W_{\text{req}} - W}{Bk/\text{EP}}, \qquad \eta = 1 - \frac{W}{W_{\max}}. \tag{35}$$

In this EP8 study, $Bk/\mathrm{EP} = 6bs$. As in Appendix D, receive-buffer savings in MiB count only BF16 vectors of width 2048: $(W_{\max} - W)/256$, excluding GEMM buffers, fixed overhead and the memory occupied by additional expert weights.

E.4 Results

Figures 9 and 10 compare all four replica counts under *Adjacent* and *Interleaved* dispatch. Without additional replicas, the rules coincide. With 16 or 32 additional replicas, the planned capacities still agree between the rules across all tested configurations. In these configurations, each replicated expert has two copies, and the two source mappings differ by a common permutation of source ranks. With equal source batch sizes and identical calibration marginals, that permutation leaves the MGF sum in (34) unchanged. However, it does not necessarily preserve the loads of contiguous prefill batches, whose source groups contain different token sequences.

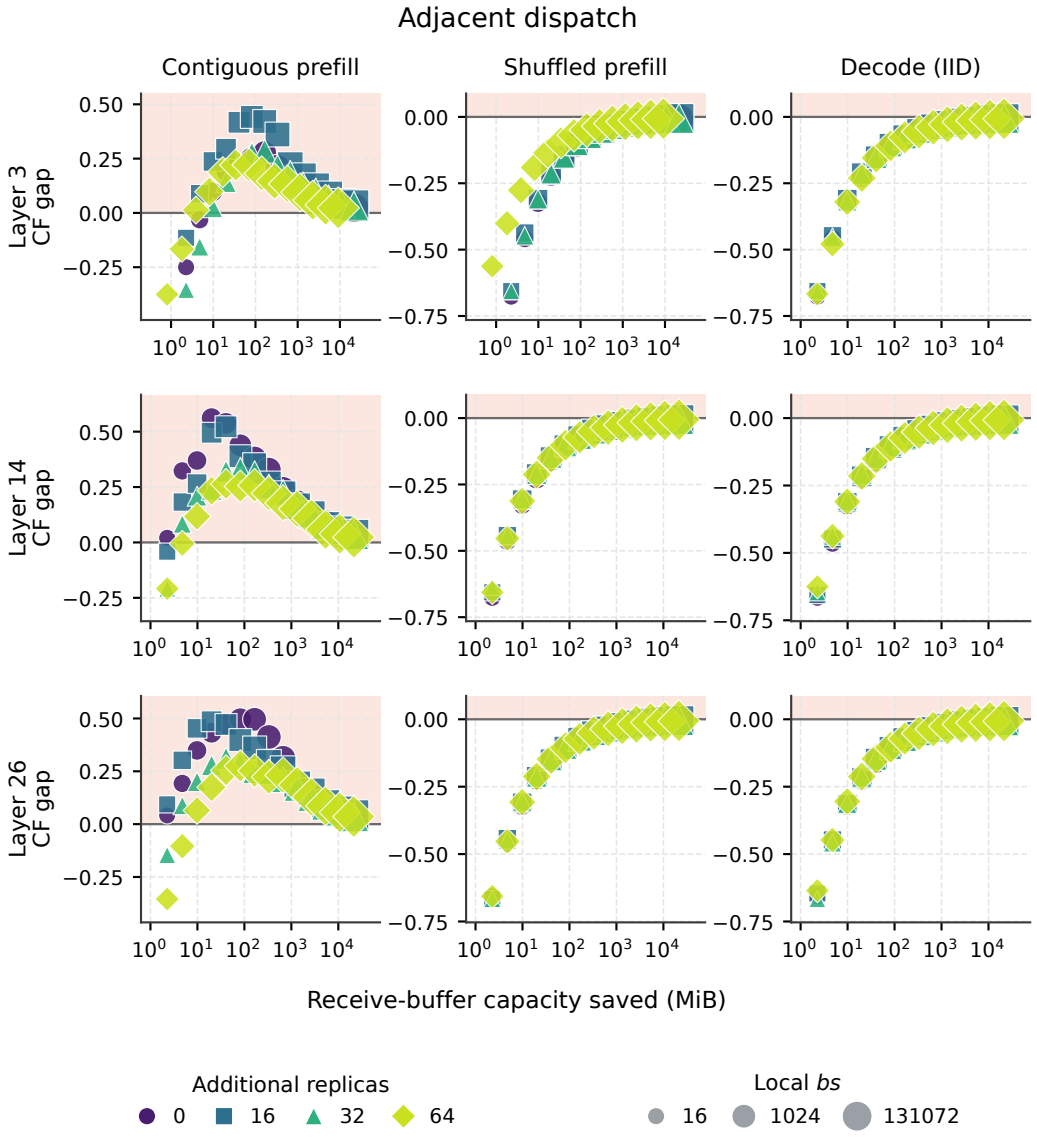


Figure 9: **Effects of expert replication under *Adjacent* dispatch.** DeepSeek, EP8, all three captured layers and three traffic settings. Color and marker shape identify 0, 16, 32 or 64 additional replicas; marker size denotes local bs. The horizontal axis counts BF16 receive-buffer capacity saved against the configuration's full reservation. Positive CF gaps are shaded. All 504 configurations are shown.

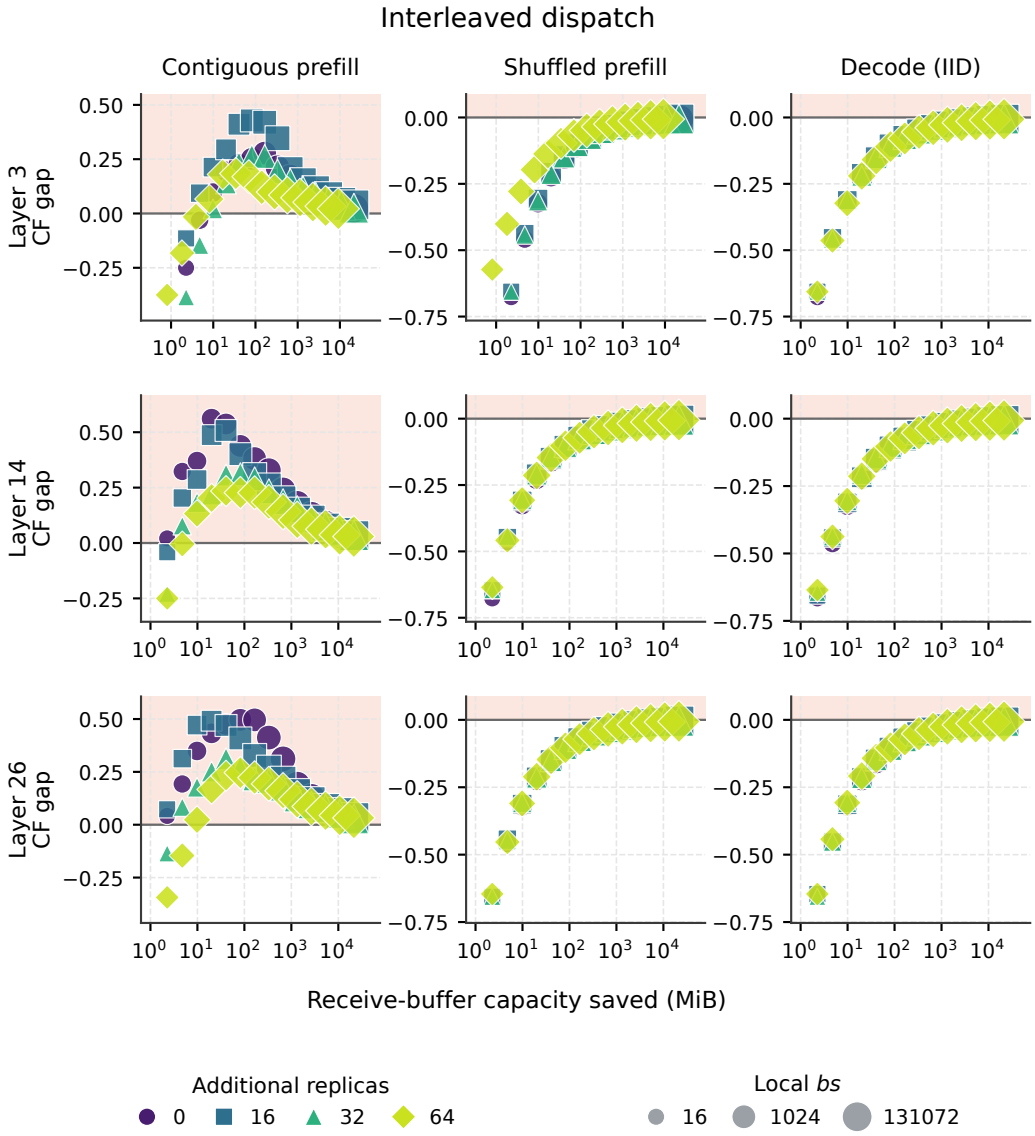


Figure 10: **Effects of expert replication under *Interleaved* dispatch.** The 504 configurations match Figure 9 in logical routing samples, replica counts and physical placements. Corresponding panels use identical axis limits and encodings.

Table 9 gives selected capacity factors and gaps for layer 14. At local batch size 8192, the *Adjacent* prefill capacity changes from 51692 rows with no additional replicas to 51086, 51025 and 51774 rows with 16, 32 and 64 additional replicas. Thus, more replicas do not monotonically reduce the planned capacity. With 64 additional replicas, *Interleaved* plans 51776 rows, only two more than *Adjacent*, but its empirical contiguous-prefill requirement is 56889 rather than 59213 rows. The corresponding CF gaps are 0.10402 and 0.15135. With 64 additional replicas, planned capacities differ in only 34 of the 126 tested configurations, by -27 to $+3$ rows for *Interleaved* minus *Adjacent*.

Table 9: Capacity factors and CF gaps for DeepSeek layer 14, EP8. Prefill CF is shared by contiguous and shuffled prefill. Additional replicas exclude the original 64 experts. Positive gaps indicate observed overflow above 1%.

Dispatch	Local bs	Additional replicas	Prefill CF	Contig. CF gap	Shuffled CF gap	Decode CF	Decode CF gap
Adjacent	128	0	1.32943	+0.56120	−0.22786	1.33073	−0.22396
Adjacent	128	16	1.31510	+0.49479	−0.21094	1.31510	−0.21615
Adjacent	128	32	1.31380	+0.23698	−0.21094	1.30859	−0.20833
Adjacent	128	64	1.31641	+0.23307	−0.21224	1.32161	−0.21484
Adjacent	8192	0	1.05168	+0.17716	−0.02698	1.05343	−0.02724
Adjacent	8192	16	1.03935	+0.16349	−0.02555	1.03986	−0.02576
Adjacent	8192	32	1.03811	+0.17828	−0.02490	1.03866	−0.02425
Adjacent	8192	64	1.05334	+0.15135	−0.02553	1.05711	−0.02612
Interleaved	128	0	1.32943	+0.56120	−0.22786	1.33073	−0.22396
Interleaved	128	16	1.31510	+0.48698	−0.20964	1.31510	−0.21745
Interleaved	128	32	1.31380	+0.21875	−0.21094	1.30859	−0.20573
Interleaved	128	64	1.31641	+0.20312	−0.21354	1.31901	−0.21354
Interleaved	8192	0	1.05168	+0.17716	−0.02698	1.05343	−0.02724
Interleaved	8192	16	1.03935	+0.14661	−0.02570	1.03986	−0.02586
Interleaved	8192	32	1.03811	+0.15326	−0.02551	1.03866	−0.02405
Interleaved	8192	64	1.05339	+0.10402	−0.02531	1.05697	−0.02570

For layer 3 prefill with 64 additional replicas, full reservation is $24bs$ rather than the $48bs$ used by the other configurations. For example, at $bs = 8192$ under *Adjacent* dispatch, the planned capacity decreases from 51315 rows without replication to 50969 rows with 64 additional replicas, but the compression rate decreases from 86.95% to 74.08% because full reservation has halved.

All 168 shuffled-prefill and all 168 decode configurations for each rule have zero observed overflow. Contiguous prefill remains different: 154/168 *Adjacent* and 153/168 *Interleaved* configurations exceed the 1% budget, with maximum overflow of 94.40% and 95.69%. Even the layer-14, $bs = 8192$, 64-replica example above has overflow of 68.65% and 63.25%, respectively.

F Experimental setup for Figure 1(a)

We use Qwen L19 with EP8 and global batch size 512. From the routing records of 131072 tokens, we identify experts that tend to be selected together and place them on the same card, while keeping the number of experts equal across cards. We fix this placement and use a separate set of 131072 tokens for evaluation. We then sample 200000 batches with replacement from these evaluation records, preserving each token's complete expert selections. This placement is chosen to illustrate the effect of expert co-selection.

For this comparison, both MarCap and the independent-expert approximation use the marginals computed from the evaluation records and the same expert placement. At a risk budget of 1%, the independent-expert approximation gives a capacity of 1173 assignment rows per card. Of the 200000 sampled batches, 3022 overflow at this capacity, giving an observed overflow frequency of 1.511%, while the independent-expert approximation predicts only 0.01096%. MarCap gives a capacity of 1355 assignment rows per card, with no observed overflow in the same batches.

G Routing experiments with 25% calibration

G.1 Calibration and batch construction

Both models are given the identical 837383 WildChat requests. A request-level random permutation assigns 209345 requests to calibration and the remaining 628038 to evaluation. For each stage and layer, an expert's marginal is computed by its selection count divided by the number of calibration tokens. LPT based placement assigns experts in descending marginal order to the least-loaded eligible

rank, with an equal number of experts per rank. The capacities and expert placement are computed using calibration data only.

Contiguous prefill preserves within-request token order, while shuffled prefill permutes the same evaluation token population. A common aligned prefix ensures complete batches at every tested size. The retained prefill populations contain approximately 568 million Qwen and 671 million DeepSeek tokens. IID-position decode forms 10000 batches at each fixed batch size by sampling complete token routes independently with replacement from the held-out decode population. These fixed-size tests preserve within-token expert choices but do not replay request arrivals, where request-level scheduling is described in Appendix I.

When computing $W_{\mathrm{req}}(\varepsilon)$, we allow at most $\lfloor \varepsilon n \rfloor$ overflows among n evaluated batches.

G.2 Overflow in shuffled prefill and decode

For the largest DeepSeek shuffled-prefill batch size in Table 10, the evaluation data provide only 80 batches. Even one overflowing batch would give an overflow frequency of 1.25%, so meeting the 1% budget requires no observed overflow.

Table 10: All shuffled-prefill and decode configurations with observed overflow frequency above 1%. The CF gap is defined in (8); n is the number of evaluated batches.

Model	Layer	Setting	EP	Global B	ΔCF	Overflows/n	Overflow %
DeepSeek	26	Decode	64	524288	0.000427	130/10000	1.30
DeepSeek	26	Decode	32	1048576	0.000137	129/10000	1.29
DeepSeek	26	Decode	64	1048576	0.003784	871/10000	8.71
DeepSeek	26	Decode	32	2097152	0.002103	1742/10000	17.42
DeepSeek	26	Decode	64	2097152	0.005798	4365/10000	43.65
DeepSeek	26	Decode	32	4194304	0.003539	7826/10000	78.26
DeepSeek	26	Decode	64	4194304	0.007439	9373/10000	93.73
DeepSeek	26	Decode	64	8388608	0.008555	9999/10000	99.99
DeepSeek	14	Shuffled prefill	64	4194304	0.000888	4/160	2.50
DeepSeek	14	Shuffled prefill	64	8388608	0.001952	21/80	26.25
Qwen	39	Decode	256	262144	0.000244	104/10000	1.04
Qwen	39	Decode	128	524288	0.006561	751/10000	7.51
Qwen	39	Decode	256	524288	0.013916	825/10000	8.25
Qwen	39	Decode	128	1048576	0.011475	4423/10000	44.23
Qwen	39	Decode	256	1048576	0.022949	4528/10000	45.28
Qwen	39	Decode	256	2097152	0.029434	9554/10000	95.54

G.3 Receive-buffer compression and CF gaps across layers

Figure 11 reports receive-buffer compression for the middle layers, while Figures 12 and 13 extend Figure 2 to the other tested layers.

The estimated capacity-dependent workspace savings per rank in Figures 2, 12 and 13 are

$$\text{Saved workspace (MiB)} = \frac{(W_{\max} - W_{\mathrm{mar}})\, b_{\mathrm{slot}}}{2^{20}}, \tag{36}$$

Here b_{slot} counts the bytes per assignment row in the capacity-dependent buffers. We assume BF16 data, with two bytes per element. The two GEMMs share the same output buffer, and the received token vectors share another buffer with the activation output. Each buffer must accommodate the larger of the two row widths, so we have

$$b_{\mathrm{slot}} = 2\big[\max(2d_{\mathrm{ff}}, d_{\mathrm{model}}) + \max(d_{\mathrm{model}}, d_{\mathrm{ff}})\big], \tag{37}$$

where d_{model} and d_{ff} are the hidden and intermediate widths. Qwen uses $(d_{\mathrm{model}}, d_{\mathrm{ff}}) = (2048, 512)$, giving $b_{\mathrm{slot}} = 8192$ bytes; DeepSeek uses $(2048, 1408)$, giving $b_{\mathrm{slot}} = 9728$ bytes. These estimates exclude fixed allocation overhead. Measurements of total operator scratch are reported separately in Appendix J.

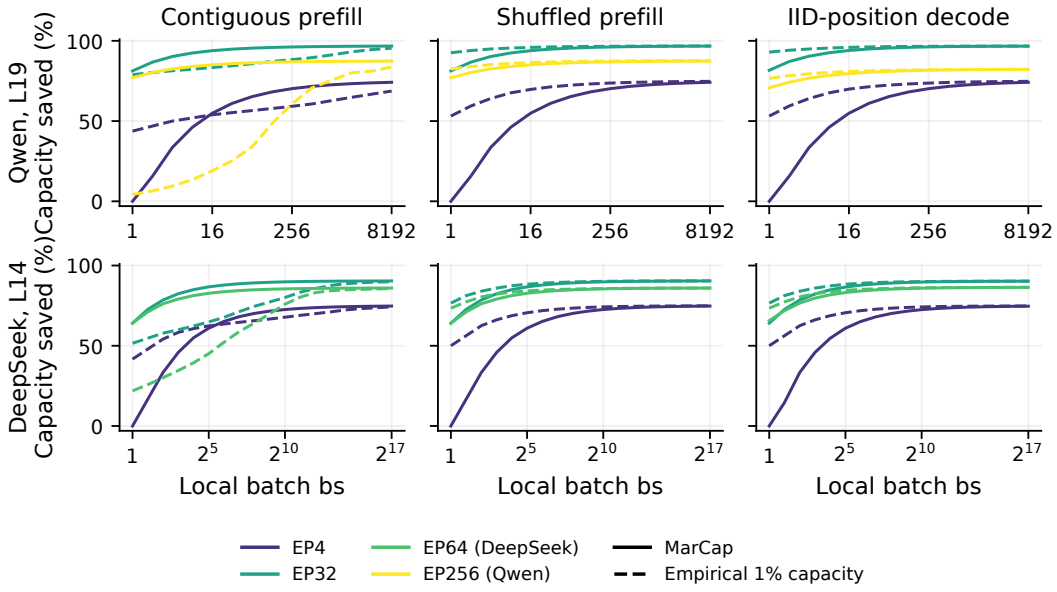


Figure 11: **Receive-buffer compression relative to full reservation.** Results for Qwen L19 and DeepSeek L14 at three EP settings. Solid lines use MarCap's capacity W_{mar}, while dashed lines use the empirical capacity $W_{\mathrm{req}}(0.01)$.

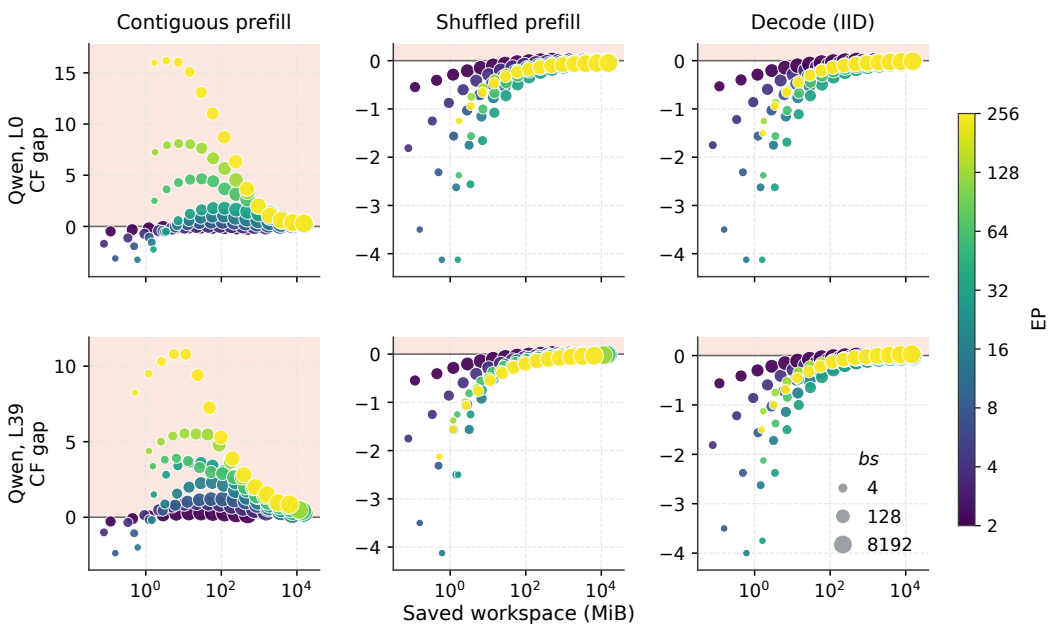


Figure 12: **CF gaps and estimated workspace savings for Qwen layers 0 and 39.** Columns show contiguous prefill, shuffled prefill and decode. The x-axis shows estimated capacity-dependent workspace savings in MiB on a logarithmic scale; the y-axis shows the CF gap, with a separate linear range for each plot. Color denotes EP and marker size denotes local bs. Configurations with zero savings are omitted from the logarithmic axis.

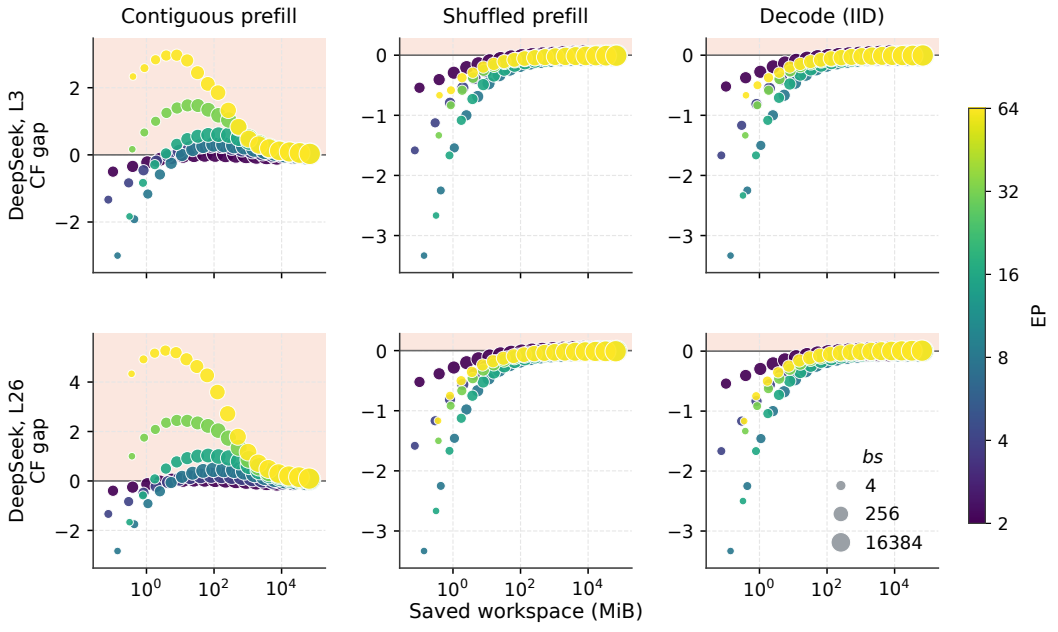


Figure 13: **CF gaps and estimated workspace savings for DeepSeek layers 3 and 26.** Columns, axes, colors and marker sizes are as in Figure 12.

G.4 QUANTIFYING TEMPORAL EFFECTS

We quantify temporal effects using the CF gap for contiguous prefill minus that for shuffled prefill (Figures 14–19). Shuffling changes which tokens appear together in a batch without changing their expert selections, so it does not remove the marginal difference between the calibration and evaluation data.

The largest temporal effects occur at large EP and small-to-moderate local batch sizes. At EP64, the CF-gap difference reaches 8.38 for Qwen L19 and 5.50 for DeepSeek L14 at $bs = 1$, but falls to 0.60 and 0.17, respectively, at $bs = 8192$. At EP256, Qwen L19 peaks at 25.00 at $bs = 1$ and still has a difference of 12.44 at $bs = 128$, while Qwen L39 peaks at 11.56 at $bs = 8$.

We conjecture that these peaks partly reflect routing behavior learned during training, and that stronger expert specialization may contribute to Qwen's larger peaks. Qiu et al. (2025) show that computing the auxiliary load-balancing loss over a global batch rather than each micro-batch relaxes the pressure for balance within individual sequences and encourages domain specialization. The original Qwen3 MoE models adopt this global-batch objective (Qwen Team, 2025), although the report does not establish the training loss of the Qwen3.6 checkpoint evaluated here. If consecutive tokens favor the same specialized experts, a small processing batch can be highly imbalanced even when loads balance over many requests. With fewer experts on each card at large EP, this concentration could pose greater challenges for inference-time load balancing and receive-buffer provisioning.

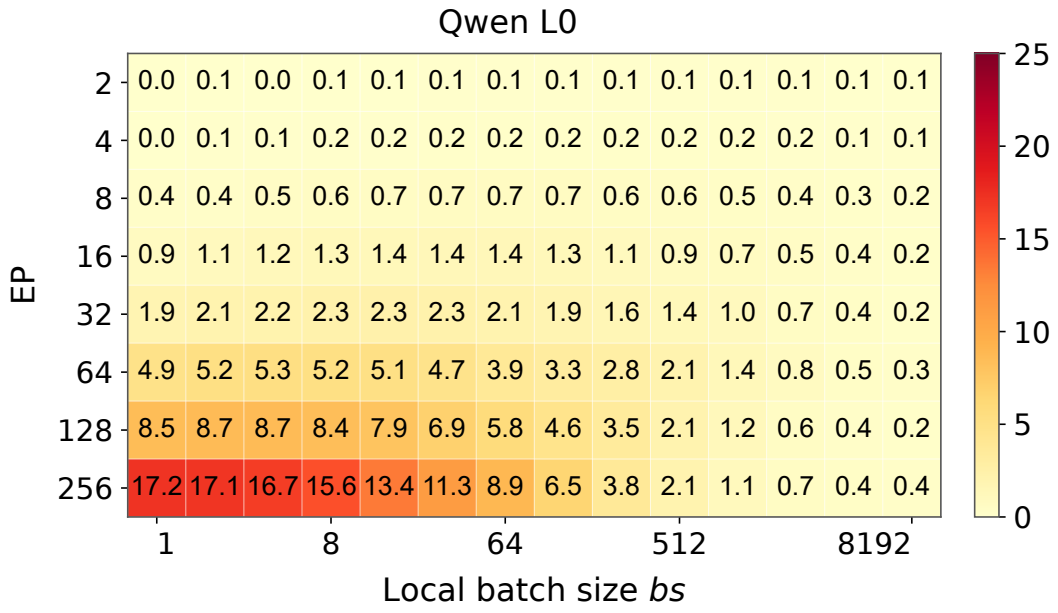


Figure 14: **Effect of shuffling on required capacity: Qwen L0.** Colors show the CF gap for contiguous prefill minus that for shuffled prefill at $\varepsilon = 0.01$, using the same tokens, expert placement and MarCap capacity. Cell values are rounded to one decimal place. The three Qwen layers share a linear color scale from 0 to 25.

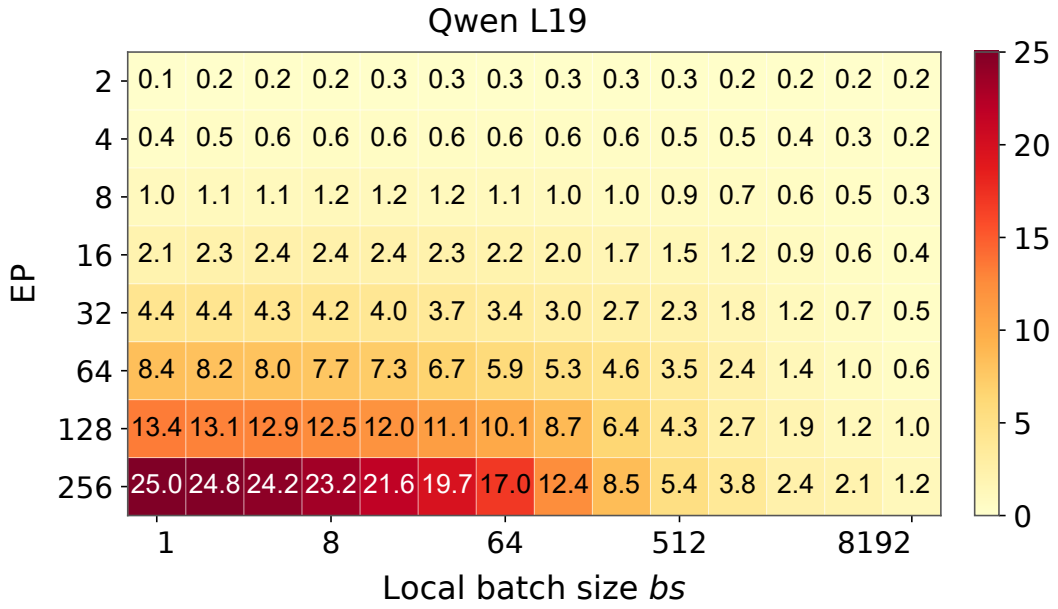


Figure 15: **Effect of shuffling: Qwen L19.** The comparison and color scale follow Figure 14.

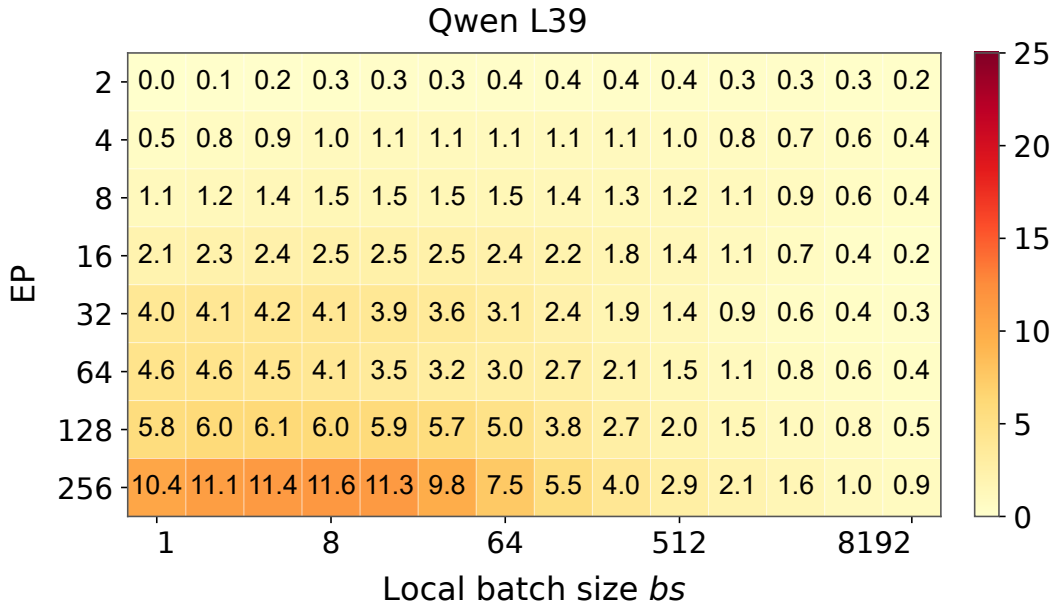


Figure 16: **Effect of shuffling: Qwen L39.** The comparison and color scale follow Figure 14.

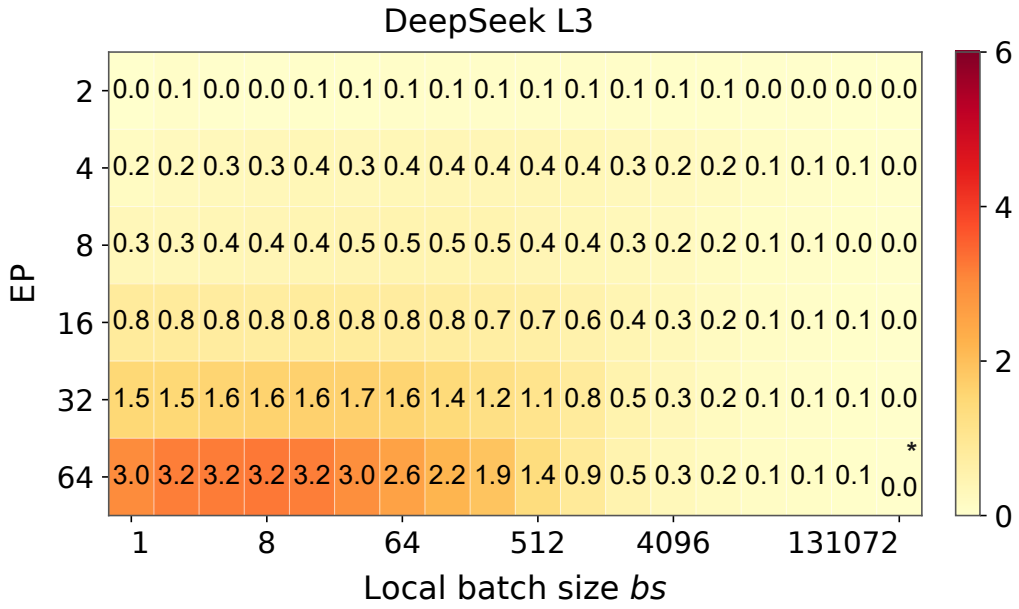


Figure 17: **Effect of shuffling: DeepSeek L3.** The comparison follows Figure 14, with a linear color scale from 0 to 6 shared by the three DeepSeek layers. Asterisks mark configurations with only 80 batches, where W_{req} equals the largest observed card load.

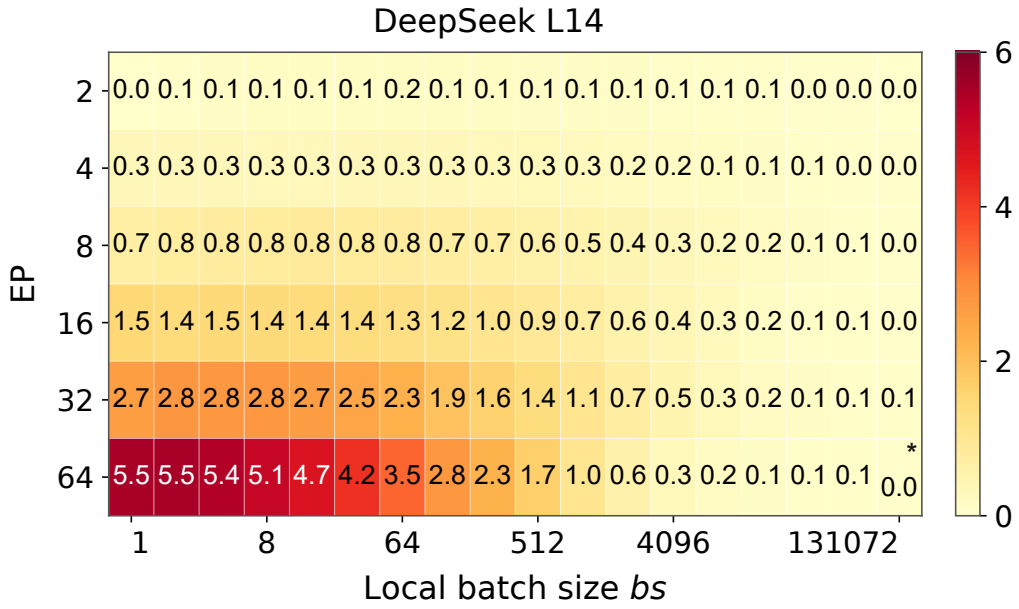


Figure 18: **Effect of shuffling: DeepSeek L14.** The comparison, color scale and asterisk notation follow Figure 17.

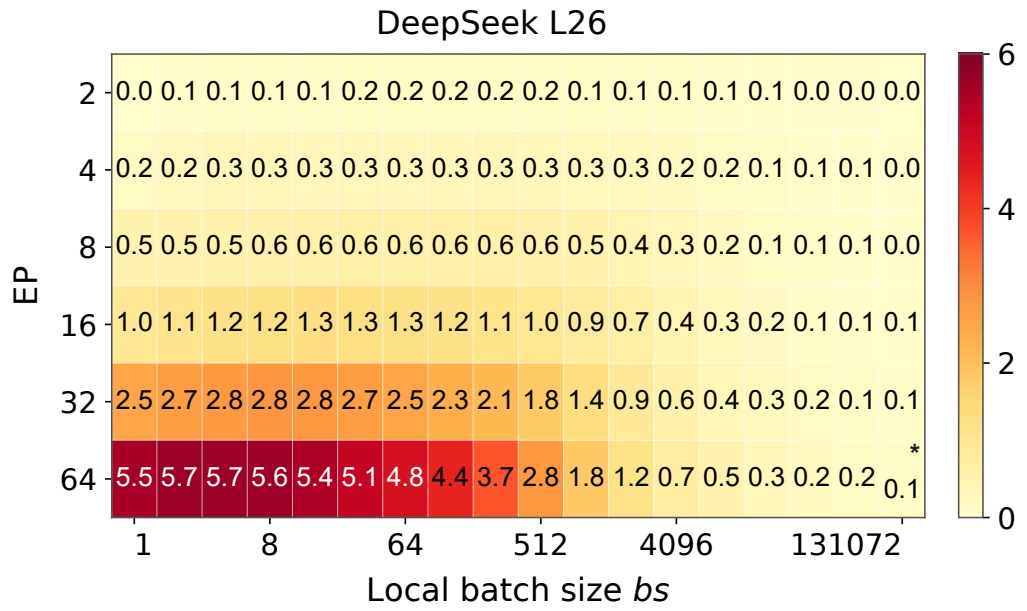


Figure 19: **Effect of shuffling: DeepSeek L26.** The comparison, color scale and asterisk notation follow Figure 17.

H Robustness across risk budgets

We use the same calibration and evaluation data as in Appendix G, keeping the marginals and expert placement fixed while varying the risk budget. The batch-size and EP settings are DeepSeek $(bs, \mathrm{EP}) \in \{(512, 8), (256, 16)\}$ and Qwen $(bs, \mathrm{EP}) \in \{(128, 16), (64, 32)\}$. Within each model, these settings have the same global batch size B but different expert placements. We evaluate all three layers for shuffled prefill and decode at $\varepsilon \in \{10^{-2}, 10^{-4}, 10^{-6}\}$, totaling 72 configurations.

Prefill uses the same shuffled batches as Appendix G. For decode, we independently sample complete token routes with replacement from the evaluation data, using ten million batches per configuration at every risk budget. The two batch-size and EP settings within a layer use the same sampled token records, and all budgets are evaluated on the same batches. The number of evaluation batches is fixed before counting overflows.

For each budget, we recompute $W_{\mathrm{req}}(\varepsilon) = z_{(n-\lfloor \varepsilon n \rfloor)}$, where $z_{(i)}$ is the ith smallest maximum card load across the evaluated batches. We also check the numerical overflow bound at $W_{\mathrm{mar}} + 1$, i.e. the first integer load above capacity, using an LP dual solution corrected for floating-point error and high-precision exponential evaluation.

Table 11: Observed overflow frequencies at each risk budget. Each entry is the maximum over the three layers.

				Risk budget ε		
Model	Setting	(bs, EP)	n per layer	10^{-2}	10^{-4}	10^{-6}
DeepSeek	Shuffled prefill	$(512, 8)$	163840	0	0	0
DeepSeek	Decode	$(512, 8)$	10000000	0	0	0
DeepSeek	Shuffled prefill	$(256, 16)$	163840	0	0	0
DeepSeek	Decode	$(256, 16)$	10000000	0	0	0
Qwen	Shuffled prefill	$(128, 16)$	277504	0	0	0
Qwen	Decode	$(128, 16)$	10000000	0	0	0
Qwen	Shuffled prefill	$(64, 32)$	277504	0	0	0
Qwen	Decode	$(64, 32)$	10000000	0	0	0

Overflow frequency. All 36 decode comparisons have zero observed overflow in ten million batches each. A one-sided Clopper–Pearson bound with a Bonferroni correction across the 36 comparisons gives the simultaneous 95% upper bound

$$U_{95} = 1 - \left(\frac{0.05}{36}\right)^{1/10^7} = 6.579249 \times 10^{-7} < 10^{-6}. \tag{38}$$

This bound applies to independent sampling from the fixed decode evaluation data without requiring independence between the 36 comparisons. Shuffled prefill has 163840 batches per DeepSeek configuration and 277504 per Qwen configuration with no observed overflow. Even at 10^{-6} these sample counts produce no overflowing batches, making $W_{\mathrm{req}} = W_{\mathrm{zero}}$, i.e. the largest observed card load. It's worth noting that because these batches come from one shuffle rather than independent draws, we don't apply the binomial confidence bound to them.

Capacity increase. The CF gaps range from -0.49414 to -0.09375 across the 72 comparisons. Tightening the budget from 10^{-2} to 10^{-6} increases capacity by 62–267 assignment rows per card, or 4.40–16.51% relative to the corresponding 1% capacity. The compression rate decreases by 0.38–1.15 percentage points, while capacity savings remain 84.24–94.84% relative to full reservation. The CF gap doesn't necessarily change monotonically with the budget because both W_{req} and W_{mar} change.

I SLIDING-WINDOW RECALIBRATION

We compare frozen calibration, denoted by *Frozen*, which retains the initial marginals, with sliding-window recalibration, denoted by *Sliding*, which updates them from completed routing history. Both methods use the same subsequent traffic and a fixed expert placement, with risk budget $\varepsilon = 0.01$. Calibration uses the initial routing records from each run, rather than the random 25% request split in Appendix G. Figure 3 combines results from three separate experiments: prefill with Poisson arrivals at an offered load ratio of 0.75, decode with Poisson arrivals at 0.95, and full-batch decode with requests starting at different decode ages. Complete results appear in Appendix I.3.

I.1 PREFILL AND DECODE WITH POISSON ARRIVALS

Across the two models, we test 32 settings in which each request completes prefill before decoding, simulating the increasingly common practice of separating prefill and decode (Zhong et al., 2024). The scheduler processes prefill chunks in first-come, first-served order. After prefill, each active request contributes one decode token per step and may start decoding in the same step as prefill completion.

Arrival rate and offered load ratio. One simulation step allows one prefill iteration and one decode iteration, and it does not represent a fixed amount of hardware execution time. At each step, the number of new requests is drawn independently from a Poisson distribution with mean λ, in requests per step. Requests are admitted from a finite ordered list without replacement, and the final draw is capped by the number of remaining requests. We set

$$\lambda = \rho\lambda_{\mathrm{nominal}}, \qquad \lambda_{\mathrm{nominal}} = \min\left\{\frac{B_{\mathrm{p}}}{\overline{L}_{\mathrm{p}}},\ S_{\mathrm{p}},\ \frac{S_{\mathrm{d}}}{\overline{L}_{\mathrm{d}}}\right\}, \tag{39}$$

where B_{p} is the prefill token limit per step, S_{p} is the maximum number of prefill requests processed per step, and S_{d} is the active decode request limit. The means $\overline{L}_{\mathrm{p}}$ and $\overline{L}_{\mathrm{d}}$ are the recorded prompt and decode lengths of the selected requests. Thus, ρ is the arrival rate divided by the nominal request processing capacity, not the actual observed batch fill or hardware utilization. The scheduler also limits the number of requests with partially completed prefill, but this limit is not a separate term in the normalization. The prefill results in Figure 3 use $B_{\mathrm{p}} = S_{\mathrm{d}} = B$, $S_{\mathrm{p}} = 1024$ and $\rho = 0.75$, at both batch limits $B \in \{65536, 524288\}$.

The default limits are 1024 prefill requests per step and 64 requests with partially completed prefill. At global batch limit 65536, we vary the offered load ratio from 0.25 to 0.99 and both prefill limits, while at 524288 we use an offered load ratio of 0.75.

Sliding computes marginals from the previous T tokens and uses them to choose the capacity for the next H-token window. Prefill uses $H = 524288$ and $T = 4H$ while decode uses $H = T = 7340032$. Both H and T use tokens instead of token–expert assignments. The capacity is computed before observing the next window, with one assignment row added as a numerical safeguard. Both methods exclude calibration steps, the final incomplete window and any window without a capacity prediction from earlier history.

For the n evaluated steps, let $z_j = \max_c L_c^{(j)}$ and let W_j be the capacity used at step j. We compute the differences $z_j - W_j$ across all steps and take r_ε to be the smallest threshold exceeded by at most an ε fraction of them. With $\overline{W} = \frac{1}{n}\sum_{j=1}^{n} W_j$, we report

$$\Delta\mathrm{CF} = \frac{r_\varepsilon}{B_{\mathrm{ceiling}} k/\mathrm{EP}}, \qquad \eta = 1 - \frac{\overline{W}}{W_{\max}(B_{\mathrm{ceiling}})}. \tag{40}$$

Here B_{ceiling} is the configured global batch limit, not the number of active tokens, and $W_{\max}(B_{\mathrm{ceiling}})$ is full reservation at that limit. All steps have equal weight, rather than all windows. This definition preserves the relationship between a positive CF gap and an overflow count above $\lfloor \varepsilon n \rfloor$.

Effect of recalibration. For Qwen L19/EP32 at offered load ratio 0.99, *Frozen* overflows on 1224 of 12245 decode steps, while *Sliding* has no overflow on the same steps. *Sliding* has no observed decode overflow across these 32 settings. At the default offered load ratio of 0.75, *Frozen* also has no decode overflow. Batches often contain fewer active tokens than the configured limit, which helps both methods avoid overflow. Therefore, Tables 12 and 13 report mean batch fill alongside capacity savings and overflow. Recalibration does not eliminate prefill overflow, and some prefill configurations have higher overflow frequencies with *Sliding* than with *Frozen*.

Table 12: Qwen prefill and decode with Poisson arrivals. F and S denote *Frozen* and *Sliding*. Shuffled requests permutes request order at load 0.75 while preserving token order within each request. For calibration at batch limit 524288 and $\rho = 0.75$, "age zero" starts each request from the beginning of its recorded decode sequence, while "random ages" starts it at a randomly chosen position in that sequence. Both complete the required calibration window, and requests entering decode afterwards start at age zero. Prefill and decode columns show the largest overflow percentage across layers and EP settings. Fill is the mean number of active decode tokens divided by the global batch limit. Saved gives the range of *Sliding* decode capacity savings relative to full reservation. The batch limit is 65536 unless specified otherwise.

Setting	Prefill F	Prefill S	Decode F	Decode S	Fill %	Saved %
Original order, 0.75	51.08	29.45	0.00	0.00	65.1	48.8–96.4
Shuffled requests, 0.75	51.33	40.50	0.00	0.00	65.1	49.0–96.4
Load 0.25	7.87	4.36	0.00	0.00	22.5	48.7–96.5
Load 0.5	25.87	12.68	0.00	0.00	44.3	48.7–96.4
Load 0.9	80.00	44.12	0.00	0.00	76.9	48.6–96.4
Load 0.99	92.54	52.83	10.00	0.00	82.6	48.7–96.4
Partial prefill limit 1	52.29	30.42	0.00	0.00	65.1	48.8–96.4
Partial prefill limit 8	52.29	30.42	0.00	0.00	65.1	48.8–96.4
Prefill request limit 2	0.43	0.29	0.00	0.00	1.9	48.7–96.4
Prefill request limit 4	0.80	0.53	0.00	0.00	3.7	49.2–96.5
Prefill request limit 8	1.76	1.10	0.00	0.00	7.5	48.8–96.4
Prefill request limit 64	43.26	21.77	0.00	0.00	58.1	48.9–96.5
Prefill request limit 256	51.08	29.45	0.00	0.00	65.1	48.8–96.4
Prefill request limit 4096	51.08	29.45	0.00	0.00	65.1	48.8–96.4
524288; age zero	21.58	8.13	0.00	0.00	37.2	49.2–96.4
524288; random ages	21.58	8.13	0.00	0.00	37.2	48.6–96.5

Table 13: DeepSeek prefill and decode with Poisson arrivals, with the same definitions and default batch limit as Table 12.

Setting	Prefill F	Prefill S	Decode F	Decode S	Fill %	Saved %
Original order, 0.75	40.05	18.88	0.00	0.00	32.9	49.0–89.9
Shuffled requests, 0.75	28.49	25.34	0.00	0.00	32.9	49.2–89.8
Load 0.25	2.48	1.89	0.00	0.00	11.0	49.2–89.9
Load 0.5	11.20	7.29	0.00	0.00	22.0	48.6–89.9
Load 0.9	60.78	30.21	0.00	0.00	39.3	48.7–89.9
Load 0.99	78.45	41.41	0.00	0.00	42.6	48.8–89.9
Partial prefill limit 1	40.43	21.86	0.00	0.00	32.9	49.0–89.9
Partial prefill limit 8	40.43	21.85	0.00	0.00	32.9	49.0–89.9
Prefill request limit 2	0.14	0.11	0.00	0.00	1.1	49.2–90.0
Prefill request limit 4	0.27	0.20	0.00	0.00	2.2	49.2–90.0
Prefill request limit 8	0.59	0.47	0.00	0.00	4.3	49.2–89.9
Prefill request limit 64	39.73	18.96	0.00	0.00	32.9	49.0–89.9
Prefill request limit 256	40.05	18.88	0.00	0.00	32.9	49.0–89.9
Prefill request limit 4096	40.05	18.88	0.00	0.00	32.9	49.0–89.9
524288; age zero	16.26	5.83	0.00	0.00	23.4	48.4–89.9
524288; random ages	16.26	5.83	0.00	0.00	23.9	49.5–90.2

I.2 Decode with Poisson arrivals and full batches

We also run 12 experiments in which requests enter decode directly, without a prefill stage. For both models and global batch limits $B \in \{65536, 524288\}$, we study Poisson arrivals at offered load ratio 0.95 and Mixed-age full batches, with synchronous age-zero starts reported for reference. Each run uses the same 837383 recorded requests per model, excluding requests with no recorded decode tokens. Each active request contributes one token per step.

Poisson arrivals. Without prefill, the nominal request-processing capacity is $B/\overline{L}$, where B is the active decode request limit and $\overline{L}$ is the mean recorded decode length of the selected requests. The offered load ratio $\rho = 0.95$ gives an arrival rate of $\lambda = \frac{0.95B}{\overline{L}}$ requests per step. New request counts are independent Poisson draws with this mean, capped by the number of remaining requests. Requests enter at decode age zero and wait if the active request limit is reached. No request can appear twice in a batch, and the batch size equals the number of active requests.

Sliding has no observed overflow in all 84 configurations in Table 14, including the 11 where *Frozen* exceeds the 1% risk budget. The mean batch fill may be lower than the offered load ratio because fewer requests are active at the beginning and end of each run.

Table 14: Decode with Poisson arrivals at offered load ratio 0.95. Steps and mean fill are measured over the same complete evaluation windows for both methods. F $> 1\%$ counts configurations where *Frozen* exceeds the budget. Max. F/S gives the largest overflow percentage for each method, and Saved S gives the range of *Sliding* capacity savings across layers and EP settings.

Model	B	Steps	Fill %	F $> 1\%$	Max. F/S (%)	Saved S (%)
DeepSeek	65536	6376	89.97	2/18	82.89/0.00	48.92–89.68
DeepSeek	524288	1802	39.63	0/18	0.00/0.00	48.79–89.38
Qwen	65536	11398	88.80	8/24	57.70/0.00	48.49–96.35
Qwen	524288	3010	42.23	1/24	12.72/0.00	47.04–96.41

Mixed-age initialization and evaluation. For the Mixed-age full-batch tests, we select B distinct requests and start them at different decode ages, meaning different positions in their recorded decode sequences to approximate ongoing serving. Selection favors requests with longer decode sequences: for each request with positive decode length L, we draw U independently and uniformly from $(0, 1)$ and select the B requests with the smallest values of $-\frac{\log(U)}{L}$. Each selected request starts at an independently chosen uniform integer age in $\{0, \ldots, L-1\}$. The remaining requests are shuffled uniformly without length weighting. Request selection, starting ages and refill order use independent random draws.

Every replacement request starts at age zero, including during calibration. Skipped tokens are not used later, completed requests are not restarted, and batches are not padded.

Each run uses $H = T = 7340032$ decode tokens, with the same risk budget and numerical safeguard as above. Calibration determines an LPT placement that remains fixed for the run. *Sliding* updates the marginals from the preceding completed window. We form each window by accumulating complete steps until the total token count reaches or exceeds H. Both methods exclude calibration, the first window after calibration, windows without a previous capacity prediction and the incomplete final window. Full-batch results include a window only if it is complete and *every step* contains B active tokens. Other complete windows, in which batches become partly filled as requests finish, are reported separately. We use the metrics in (40) on the same evaluation steps for both methods.

Table 15: Mixed-age full-batch decoding, with Synchronous age-zero initialization shown for reference. Here n is the number of evaluated steps per configuration. The F and S columns under Above 1% count configurations exceeding the budget. Max. gives the largest overflow percentage across layers and EP settings, while Saved S gives the range of *Sliding* capacity savings relative to full reservation at B.

Model	B	Init.	n	Above 1%		Max. F/S (%)	Saved S (%)
				F	S		
DeepSeek	65536	Mixed-age	5040	0/18	0/18	0.00/0.00	49.29–90.07
DeepSeek	65536	Synchronous	5376	8/18	0/18	100.00/0.37	48.79–89.57
DeepSeek	524288	Mixed-age	154	1/18	0/18	1.95/0.00	49.62–90.36
DeepSeek	524288	Synchronous	350	12/18	9/18	100.00/13.14	48.57–89.16
Qwen	65536	Mixed-age	8848	0/24	0/24	0.00/0.00	49.32–96.59
Qwen	65536	Synchronous	9520	19/24	0/24	100.00/0.41	47.86–96.10
Qwen	524288	Mixed-age	252	5/24	0/24	74.21/0.00	49.71–96.71
Qwen	524288	Synchronous	546	20/24	17/24	100.00/10.44	46.71–96.20

Mixed-age full-batch results. *Sliding* has no observed overflow in all 84 Mixed-age configurations. Table 16 lists the six configurations where *Frozen* exceeds 1%, all at the larger batch limit. For $B = 65536$, the numbers of complete evaluation windows are 45 and 48 for DeepSeek, and 79 and 85 for Qwen, for Mixed-age and Synchronous respectively. For $B = 524288$, the corresponding counts are 11 and 25 for DeepSeek, and 18 and 39 for Qwen. Successive decoding steps can be correlated, so we do not use these results to construct an IID binomial confidence interval.

Table 16: Mixed-age configurations with *Frozen* overflow above 1%, all at $B = 524288$. *Sliding* has no observed overflow on the same n steps in every row.

Model	Layer	EP	*Frozen* overflows/n	CF gap F	CF gap S
DeepSeek	14	2	3/154	0.000397	-0.002728
Qwen	19	16	11/252	0.001011	-0.013924
Qwen	39	8	9/252	0.000877	-0.009537
Qwen	39	64	66/252	0.016769	-0.008148
Qwen	39	128	77/252	0.039215	-0.007263
Qwen	39	256	187/252	0.159546	-0.017212

For Mixed-age Qwen L39/EP256 at $B = 524288$, *Sliding* increases the mean capacity by 2.5513%, changing receive-buffer compression from 87.7649% to 87.4528%, while the observed overflow frequency falls from 74.2063% to zero.

Synchronous initialization for reference. The Synchronous setting starts all initial requests at age zero and uses the original request order for replacements. The surviving initial requests advance through decode at the same age, so age mixing relies on their completion and replacement rather than the staggered entry times of Poisson arrivals. At $B = 524288$, each H-token window spans only 14 full-batch steps, compared with 112 at $B = 65536$, and the Synchronous runs contain only 350 and 546 full-batch evaluation steps for DeepSeek and Qwen, respectively. With finite routing records, large concurrency leaves fewer steps for requests to finish and be replaced before batches stop being full, which can leave the evaluated traffic affected by the synchronized start. Therefore, we

use Mixed-age initialization to provide varied ages from the outset and retain the Synchronous case for reference. If expert selection frequencies vary with decode age, the remaining synchronization can cause rapid marginal changes between recalibrations, and *Sliding* can then exceed the risk budget (Table 15). Figure 20 compares the initializations, but the runs also differ in request selection, order, calibration marginals and expert placement, so this comparison does not isolate the starting ages alone.

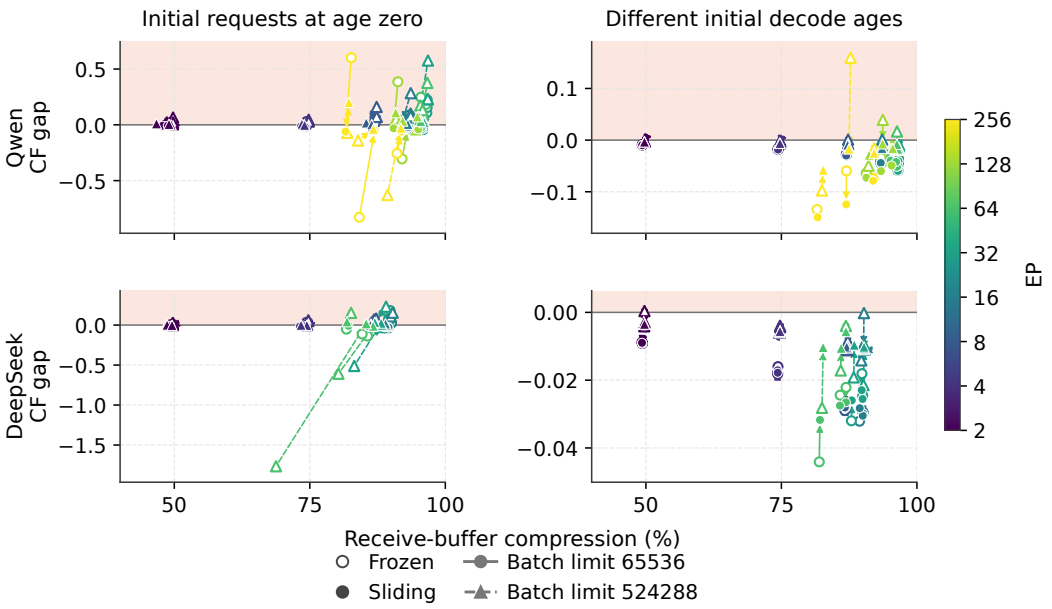


Figure 20: **Full-batch decoding with synchronous initialization for reference.** Mixed-age initialization is shown on the right, while the Synchronous setting with all initial requests starting at age zero is on the left. Arrows connect *Frozen* to *Sliding* for each configuration. All three layers and all EP settings are included at both batch limits. Axes, colors and marker sizes follow Figure 3, while each plot uses its own vertical range.

I.3 Complete results for Figure 3

The tables below include all 252 configurations, of which 48 appear in Figure 3. Prefill uses the setup in Appendix I.1; Poisson decode and full-batch decode use Appendix I.2. F and S denote *Frozen* and *Sliding*. Nonzero overflow entries give the count followed by the percentage in parentheses. Each batch-limit group lists the number of evaluated steps n per configuration, the number of complete windows and the mean batch fill. Fill is the active token count as a percentage of the batch limit; compression and CF gaps use (40).

Table 17: Qwen prefill with Poisson arrivals at offered load ratio 0.75.

Layer	EP	Compression (%)		CF gap		Overflows	
		F	S	F	S	F	S
Batch limit 65536: $n = 15373$; 1384 windows; mean fill 74.97%.							
0	2	49.19	48.71	0.030586	0.014488	1736 (11.29)	710 (4.62)
0	4	74.02	73.68	0.052315	0.032448	1787 (11.62)	905 (5.89)
0	8	86.77	86.15	0.181198	0.088547	2744 (17.85)	1079 (7.02)
0	16	92.99	92.75	0.212982	0.160492	2806 (18.25)	1564 (10.17)
0	32	96.32	96.06	0.517029	0.356628	3820 (24.85)	2158 (14.04)
0	64	96.21	95.44	1.220581	0.783447	6210 (40.40)	3108 (20.22)
0	128	95.62	94.35	2.379150	1.660400	7082 (46.07)	3985 (25.92)
0	256	92.39	90.67	3.921387	2.923340	5886 (38.29)	3719 (24.19)
19	2	48.92	48.14	0.058468	0.041801	2435 (15.84)	1210 (7.87)
19	4	73.20	72.86	0.115356	0.097664	2635 (17.14)	1966 (12.79)
19	8	86.49	85.71	0.262711	0.188828	4699 (30.57)	2379 (15.48)
19	16	92.72	92.07	0.453522	0.309967	4973 (32.35)	2651 (17.24)
19	32	96.26	95.54	0.931641	0.630188	6746 (43.88)	3669 (23.87)
19	64	95.64	94.38	1.955566	1.484863	7852 (51.08)	4528 (29.45)
19	128	92.35	90.95	3.485840	2.806152	5747 (37.38)	3940 (25.63)
19	256	84.83	82.52	6.725586	5.385254	5006 (32.56)	3631 (23.62)
39	2	48.88	47.93	0.054512	0.035683	2329 (15.15)	1065 (6.93)
39	4	73.73	72.16	0.252510	0.114868	3728 (24.25)	1685 (10.96)
39	8	86.43	85.29	0.372055	0.228653	4788 (31.15)	2364 (15.38)
39	16	91.43	91.02	0.325470	0.248230	2641 (17.18)	1694 (11.02)
39	32	91.85	91.59	0.414368	0.384521	1912 (12.44)	1385 (9.01)
39	64	84.11	83.64	1.114502	0.908813	1910 (12.42)	1519 (9.88)
39	128	68.40	67.85	1.402832	1.369629	1739 (11.31)	1426 (9.28)
39	256	36.90	35.91	2.756348	2.775879	1698 (11.05)	1417 (9.22)
Batch limit 524288: $n = 1918$; 1065 windows; mean fill 75.01%.							
0	2	49.54	49.09	0.019158	0.003084	140 (7.30)	34 (1.77)
0	4	74.47	74.14	0.039587	0.013240	117 (6.10)	48 (2.50)
0	8	87.09	86.68	0.111809	0.026897	164 (8.55)	40 (2.09)
0	16	93.33	93.03	0.162239	0.039967	198 (10.32)	63 (3.28)
0	32	96.52	96.15	0.559372	0.142738	279 (14.55)	92 (4.80)
0	64	96.35	95.70	0.921982	0.299210	364 (18.98)	121 (6.31)
0	128	95.40	94.54	1.746796	0.727753	315 (16.42)	150 (7.82)
0	256	92.11	90.99	2.772278	1.200989	259 (13.50)	150 (7.82)
19	2	49.32	48.81	0.030359	0.012706	145 (7.56)	67 (3.49)
19	4	74.05	73.35	0.069961	0.020306	199 (10.38)	70 (3.65)
19	8	86.70	86.21	0.089279	0.031309	221 (11.52)	58 (3.02)
19	16	92.56	92.46	0.191223	0.099888	142 (7.40)	102 (5.32)
19	32	96.14	95.68	0.672859	0.267952	293 (15.28)	103 (5.37)
19	64	95.63	94.58	1.331970	0.609756	396 (20.65)	155 (8.08)
19	128	93.06	91.13	2.474915	1.184143	414 (21.58)	155 (8.08)
19	256	86.25	82.91	4.456543	2.267273	368 (19.19)	156 (8.13)
39	2	48.95	48.21	0.075340	0.019654	102 (5.32)	62 (3.23)
39	4	73.47	73.20	0.061548	0.036164	104 (5.42)	70 (3.65)
39	8	86.55	85.68	0.296968	0.075850	221 (11.52)	79 (4.12)
39	16	91.60	90.87	0.643188	0.152119	169 (8.81)	65 (3.39)
39	32	92.01	91.69	0.197891	0.125557	109 (5.68)	54 (2.82)

Continued on the next page

Table 17: Qwen prefill with Poisson arrivals at offered load ratio 0.75. (continued)

Layer	EP	Compression (%)		CF gap		Overflows	
		F	S	F	S	F	S
39	64	84.38	83.91	0.363953	0.223480	91 (4.74)	59 (3.08)
39	128	68.89	68.13	0.661133	0.452667	82 (4.28)	60 (3.13)
39	256	37.89	36.49	1.298401	0.878601	76 (3.96)	60 (3.13)

Table 18: Qwen decode with Poisson arrivals at offered load ratio 0.95.

Layer	EP	Compression (%)		CF gap		Overflows	
		F	S	F	S	F	S
Batch limit 65536: $n = 11398$; 90 windows; mean fill 88.80%.							
0	2	49.36	49.28	−0.038578	−0.040630	0	0
0	4	74.41	74.02	−0.037659	−0.051338	0	0
0	8	87.01	86.66	−0.037521	−0.064682	0	0
0	16	93.37	92.86	−0.006287	−0.087341	8 (0.07)	0
0	32	96.59	96.35	−0.027039	−0.101746	0	0
0	64	96.47	96.20	−0.005127	−0.099976	34 (0.30)	0
0	128	95.44	95.28	−0.029297	−0.105713	3 (0.03)	0
0	256	91.66	91.92	−0.189941	−0.172852	0	0
19	2	49.36	48.49	−0.024731	−0.041737	0	0
19	4	74.39	73.65	−0.016884	−0.052406	0	0
19	8	87.00	86.38	−0.012314	−0.062424	0	0
19	16	93.38	92.77	0.014526	−0.083649	1943 (17.05)	0
19	32	96.62	96.18	0.082947	−0.086487	6577 (57.70)	0
19	64	95.42	94.93	0.152344	−0.091187	6219 (54.56)	0
19	128	91.23	90.63	0.208740	−0.159912	2518 (22.09)	0
19	256	82.77	81.61	0.403320	−0.317383	2479 (21.75)	0
39	2	49.35	49.09	−0.035484	−0.040180	0	0
39	4	74.40	73.51	−0.014267	−0.053345	0	0
39	8	87.01	86.51	−0.026093	−0.068756	0	0
39	16	93.38	92.83	0.018036	−0.081299	1210 (10.62)	0
39	32	96.63	96.31	0.013184	−0.093506	787 (6.90)	0
39	64	96.10	95.65	0.039429	−0.104126	3864 (33.90)	0
39	128	92.68	93.24	−0.264160	−0.118652	0	0
39	256	85.43	86.87	−0.617188	−0.229980	0	0
Batch limit 524288: $n = 3010$; 89 windows; mean fill 42.23%.							
0	2	49.77	49.48	−0.159163	−0.162792	0	0
0	4	74.79	74.26	−0.148255	−0.168701	0	0
0	8	87.32	86.84	−0.142633	−0.175974	0	0
0	16	93.61	93.14	−0.121574	−0.189262	0	0
0	32	96.73	96.41	−0.114822	−0.203041	0	0
0	64	96.59	96.11	−0.084213	−0.219879	0	0
0	128	95.42	95.09	−0.220123	−0.265625	0	0
0	256	91.33	92.15	−0.723083	−0.437439	0	0
19	2	49.77	47.04	−0.119267	−0.168831	0	0
19	4	74.79	74.34	−0.149057	−0.168725	0	0
19	8	87.31	85.82	−0.096579	−0.183151	0	0
19	16	93.62	92.53	−0.063709	−0.199944	0	0
19	32	96.79	95.53	0.100067	−0.236023	383 (12.72)	0
19	64	95.67	95.00	−0.082016	−0.261154	0	0
19	128	91.62	90.56	−0.204468	−0.493164	0	0
19	256	83.40	81.96	−0.520447	−0.958740	0	0
39	2	49.76	49.58	−0.161427	−0.162025	0	0
39	4	74.78	73.60	−0.133227	−0.171241	0	0
39	8	87.32	86.84	−0.143459	−0.175982	0	0
39	16	93.62	92.75	−0.098095	−0.191334	0	0
39	32	96.77	96.06	−0.033989	−0.212082	0	0
39	64	95.96	95.27	−0.169495	−0.236023	0	0
39	128	92.27	92.99	−0.701416	−0.358398	0	0
39	256	84.65	86.81	−1.571838	−0.674988	0	0

Table 19: Qwen full-batch decode with different initial decode ages.

Layer	EP	Compression (%)		CF gap		Overflows	
		F	S	F	S	F	S
Batch limit 65536: $n = 8848$; 79 windows; mean fill 100.00%.							
0	2	49.36	49.34	−0.008766	−0.009251	0	0
0	4	74.41	74.39	−0.016815	−0.017456	0	0
0	8	87.02	87.00	−0.027435	−0.028702	0	0
0	16	93.35	93.35	−0.044891	−0.044830	0	0
0	32	96.50	96.50	−0.058472	−0.059509	0	0
0	64	96.30	96.31	−0.052612	−0.051758	0	0
0	128	95.35	95.34	−0.045654	−0.049072	0	0
0	256	91.92	91.92	−0.074707	−0.078613	0	0
19	2	49.36	49.33	−0.009373	−0.009892	0	0
19	4	74.38	74.37	−0.017899	−0.018234	0	0
19	8	87.00	86.98	−0.028595	−0.030060	0	0
19	16	93.39	93.35	−0.038452	−0.044189	0	0
19	32	96.60	96.59	−0.035950	−0.040955	0	0
19	64	95.13	95.15	−0.036499	−0.039673	0	0
19	128	90.64	90.68	−0.063965	−0.072754	0	0
19	256	81.58	81.68	−0.134277	−0.149414	0	0
39	2	49.35	49.32	−0.009056	−0.009789	0	0
39	4	74.40	74.38	−0.017334	−0.018227	0	0
39	8	86.99	86.96	−0.026886	−0.029861	0	0
39	16	93.35	93.33	−0.041077	−0.044281	0	0
39	32	96.60	96.58	−0.045715	−0.050293	0	0
39	64	96.15	96.15	−0.033081	−0.042358	0	0
39	128	93.39	93.37	−0.034180	−0.060059	0	0
39	256	87.01	86.96	−0.059570	−0.124512	0	0
Batch limit 524288: $n = 252$; 18 windows; mean fill 100.00%.							
0	2	49.77	49.75	−0.002906	−0.003301	0	0
0	4	74.79	74.78	−0.006217	−0.006400	0	0
0	8	87.33	87.31	−0.007957	−0.010246	0	0
0	16	93.60	93.59	−0.013374	−0.015526	0	0
0	32	96.67	96.66	−0.013046	−0.019524	0	0
0	64	96.48	96.48	−0.019958	−0.017548	0	0
0	128	95.57	95.56	−0.014801	−0.016357	0	0
0	256	92.15	92.12	−0.015808	−0.025208	0	0
19	2	49.77	49.71	−0.001747	−0.003028	0	0
19	4	74.78	74.68	−0.000864	−0.005651	0	0
19	8	87.29	87.26	−0.007687	−0.010431	0	0
19	16	93.60	93.53	0.001011	−0.013924	11 (4.37)	0
19	32	96.71	96.71	−0.010147	−0.014114	0	0
19	64	95.37	95.42	−0.025482	−0.015991	0	0
19	128	91.08	91.19	−0.049652	−0.028931	0	0
19	256	82.48	82.71	−0.097534	−0.057983	0	0
39	2	49.76	49.74	−0.003036	−0.003546	0	0
39	4	74.79	74.74	−0.003308	−0.006064	0	0
39	8	87.31	87.22	0.000877	−0.009537	9 (3.57)	0
39	16	93.60	93.58	−0.010353	−0.015816	0	0
39	32	96.73	96.71	−0.010017	−0.017677	0	0
39	64	96.33	96.31	0.016769	−0.008148	66 (26.19)	0
39	128	93.71	93.67	0.039215	−0.007263	77 (30.56)	0
39	256	87.76	87.45	0.159546	−0.017212	187 (74.21)	0

Table 20: DeepSeek prefill with Poisson arrivals at offered load ratio 0.75.

Layer	EP	Compression (%)		CF gap		Overflows	
		F	S	F	S	F	S
Batch limit 65536: $n = 18267$; 1645 windows; mean fill 74.99%.							
3	2	49.16	48.73	0.018270	0.010091	1859 (10.18)	667 (3.65)
3	4	73.92	73.63	0.033895	0.025533	1471 (8.05)	839 (4.59)
3	8	86.73	86.39	0.080404	0.051310	2795 (15.30)	1081 (5.92)
3	16	89.78	89.28	0.195435	0.138957	4580 (25.07)	1829 (10.01)
3	32	89.55	88.72	0.385417	0.292318	6419 (35.14)	3114 (17.05)
3	64	85.92	85.85	0.503743	0.500488	2949 (16.14)	2816 (15.42)
14	2	49.23	48.73	0.027568	0.017156	2563 (14.03)	1090 (5.97)
14	4	73.84	73.73	0.037669	0.032654	1632 (8.93)	1272 (6.96)
14	8	86.38	86.10	0.104065	0.078227	2866 (15.69)	1582 (8.66)
14	16	89.53	89.05	0.252075	0.177205	4524 (24.77)	2325 (12.73)
14	32	89.47	88.13	0.536377	0.364014	7316 (40.05)	3449 (18.88)
14	64	84.92	84.47	0.762370	0.672201	4034 (22.08)	3211 (17.58)
26	2	48.88	48.56	0.034551	0.021317	1903 (10.42)	1058 (5.79)
26	4	74.13	73.54	0.075226	0.047496	3539 (19.37)	1429 (7.82)
26	8	86.65	86.13	0.158976	0.097758	3960 (21.68)	1801 (9.86)
26	16	89.32	88.63	0.339722	0.251465	4882 (26.73)	2516 (13.77)
26	32	87.63	86.40	0.755290	0.573812	5009 (27.42)	2959 (16.20)
26	64	81.94	80.13	1.504557	1.173340	4433 (24.27)	3075 (16.83)
Batch limit 524288: $n = 2281$; 1279 windows; mean fill 74.99%.							
3	2	49.59	48.79	0.026854	0.004988	263 (11.53)	53 (2.32)
3	4	74.38	74.04	0.064392	0.010330	140 (6.14)	51 (2.24)
3	8	87.08	86.74	0.058609	0.014613	243 (10.65)	55 (2.41)
3	16	89.86	89.57	0.110443	0.061879	199 (8.72)	81 (3.55)
3	32	89.83	89.07	0.226481	0.086965	365 (16.00)	94 (4.12)
3	64	87.04	86.25	0.271891	0.122477	236 (10.35)	87 (3.81)
14	2	49.46	49.10	0.011641	0.003754	153 (6.71)	49 (2.15)
14	4	74.49	74.08	0.039814	0.012174	239 (10.48)	66 (2.89)
14	8	87.09	86.57	0.067589	0.020335	323 (14.16)	63 (2.76)
14	16	89.95	89.53	0.127324	0.049815	317 (13.90)	80 (3.51)
14	32	89.58	88.70	0.264313	0.110036	371 (16.26)	114 (5.00)
14	64	84.76	84.86	0.245422	0.155762	133 (5.83)	94 (4.12)
26	2	49.45	49.01	0.028182	0.006800	160 (7.01)	65 (2.85)
26	4	74.39	73.87	0.064428	0.021809	208 (9.12)	66 (2.89)
26	8	86.65	86.47	0.097580	0.031288	146 (6.40)	79 (3.46)
26	16	89.31	89.00	0.184764	0.072083	165 (7.23)	93 (4.08)
26	32	87.41	86.74	0.491211	0.188385	222 (9.73)	117 (5.13)
26	64	81.38	80.61	0.964803	0.420756	214 (9.38)	133 (5.83)

Table 21: DeepSeek decode with Poisson arrivals at offered load ratio 0.95.

Layer	EP	Compression (%)		CF gap		Overflows	
		F	S	F	S	F	S
Batch limit 65536: $n = 6376$; 51 windows; mean fill 89.97%.							
3	2	49.34	48.92	−0.043513	−0.052124	0	0
3	4	74.34	74.14	−0.053009	−0.062632	0	0
3	8	86.94	86.63	−0.050720	−0.074443	0	0
3	16	90.05	89.68	−0.043620	−0.081543	0	0
3	32	89.87	89.00	0.007161	−0.085124	309 (4.85)	0
3	64	86.01	86.96	−0.198730	−0.091471	0	0
14	2	49.34	48.97	−0.046021	−0.052434	0	0
14	4	74.33	73.93	−0.046031	−0.063243	0	0
14	8	86.87	86.76	−0.068400	−0.074483	0	0
14	16	89.88	89.40	−0.031535	−0.082723	0	0
14	32	89.86	88.25	0.085938	−0.084798	5285 (82.89)	0
14	64	85.86	85.73	−0.084635	−0.092936	0	0
26	2	49.29	49.25	−0.051214	−0.052633	0	0
26	4	74.35	74.16	−0.053569	−0.061829	0	0
26	8	86.68	86.67	−0.074849	−0.074951	0	0
26	16	89.40	89.51	−0.089233	−0.078898	0	0
26	32	87.72	87.97	−0.099284	−0.080566	0	0
26	64	81.44	82.11	−0.173340	−0.111816	0	0
Batch limit 524288: $n = 1802$; 50 windows; mean fill 39.63%.							
3	2	49.75	49.12	−0.154811	−0.166715	0	0
3	4	74.76	73.91	−0.146717	−0.173571	0	0
3	8	87.26	86.40	−0.130936	−0.183121	0	0
3	16	90.38	89.22	−0.088338	−0.194438	0	0
3	32	89.96	89.00	−0.142049	−0.193970	0	0
3	64	85.87	86.64	−0.363688	−0.241435	0	0
14	2	49.74	49.29	−0.155378	−0.167033	0	0
14	4	74.68	72.75	−0.100175	−0.183483	0	0
14	8	87.23	86.15	−0.130363	−0.185193	0	0
14	16	90.15	88.75	−0.070969	−0.204534	0	0
14	32	89.23	88.24	−0.153870	−0.204915	0	0
14	64	82.55	85.22	−0.631124	−0.254517	0	0
26	2	49.71	48.79	−0.148062	−0.168369	0	0
26	4	74.76	73.54	−0.135740	−0.175817	0	0
26	8	87.09	86.59	−0.159149	−0.180982	0	0
26	16	89.82	89.38	−0.190918	−0.188039	0	0
26	32	88.29	88.18	−0.207428	−0.215159	0	0
26	64	82.17	82.86	−0.340190	−0.321208	0	0

Table 22: DeepSeek full-batch decode with different initial decode ages.

Layer	EP	Compression (%)		CF gap		Overflows	
		F	S	F	S	F	S
Batch limit 65536: $n = 5040$; 45 windows; mean fill 100.00%.							
3	2	49.36	49.35	−0.009069	−0.009171	0	0
3	4	74.37	74.34	−0.016001	−0.017181	0	0
3	8	86.94	86.92	−0.026286	−0.027771	0	0
3	16	90.07	90.07	−0.029541	−0.029785	0	0
3	32	90.00	89.99	−0.024577	−0.025553	0	0
3	64	86.92	86.90	−0.022135	−0.026530	0	0
14	2	49.35	49.35	−0.009191	−0.009216	0	0
14	4	74.29	74.31	−0.017710	−0.017090	0	0
14	8	86.83	86.83	−0.028361	−0.028198	0	0
14	16	90.01	90.00	−0.029297	−0.030477	0	0
14	32	89.86	89.82	−0.018066	−0.022949	0	0
14	64	85.84	85.80	−0.024414	−0.027507	0	0
26	2	49.29	49.29	−0.008804	−0.008972	0	0
26	4	74.32	74.33	−0.018199	−0.017812	0	0
26	8	86.70	86.71	−0.028971	−0.028056	0	0
26	16	89.42	89.46	−0.032104	−0.028280	0	0
26	32	87.92	87.99	−0.031982	−0.025960	0	0
26	64	81.99	82.11	−0.044108	−0.031738	0	0
Batch limit 524288: $n = 154$; 11 windows; mean fill 100.00%.							
3	2	49.76	49.75	−0.002939	−0.003458	0	0
3	4	74.75	74.70	−0.003944	−0.005980	0	0
3	8	87.25	87.23	−0.008471	−0.009768	0	0
3	16	90.36	90.36	−0.011007	−0.011007	0	0
3	32	90.27	90.26	−0.009532	−0.010661	0	0
3	64	86.90	86.91	−0.003988	−0.005656	0	0
14	2	49.74	49.62	0.000397	−0.002728	3 (1.95)	0
14	4	74.68	74.70	−0.006012	−0.005950	0	0
14	8	87.21	87.22	−0.011175	−0.010498	0	0
14	16	90.23	90.17	−0.000203	−0.010737	0	0
14	32	90.14	90.25	−0.021434	−0.010223	0	0
14	64	85.95	86.02	−0.017131	−0.010559	0	0
26	2	49.70	49.74	−0.004103	−0.003466	0	0
26	4	74.78	74.75	−0.004346	−0.005578	0	0
26	8	87.04	87.05	−0.010689	−0.010190	0	0
26	16	89.75	89.80	−0.014150	−0.009618	0	0
26	32	88.34	88.45	−0.019175	−0.009644	0	0
26	64	82.50	82.70	−0.028158	−0.010335	0	0

J MegaMoE Replay: implementation and hardware experiments

J.1 Capacity and buffers in existing MoE systems

Tutel (Hwang et al., 2023) and DeepSpeed-MoE (Rajbhandari et al., 2022) expose capacity factors for controlling per-expert capacity, and both support execution without dropping assignments. Tutel's zero capacity factor selects capacity from the required expert loads, while DeepSpeed's no-drop gating paths can increase capacity according to realized expert counts.[1] Therefore, dropless execution does not necessarily require reserving the structural worst case, but choosing capacity from realized counts differs from choosing receive capacity before observing the current batch's routing.

Existing implementations also control intermediate buffers during expert computation. In vLLM (Kwon et al., 2023), the v0.10.2 fused-expert implementation processes input tokens in chunks while retaining their selected experts, reuses intermediate storage across chunks and shares storage between two intermediate tensors.[2] SGLang (Zheng et al., 2024) provides an FP8 CUTLASS implementation that allocates intermediates for the routed assignments and accepts a reusable workspace for grouped matrix multiplication.[3] These examples concern specific computation backends, whose intermediate buffers are distinct from the receive buffers used for expert-parallel communication.

MarCap addresses the choice of per-rank receive capacity from expert marginals and an overflow budget before the current routing is observed. Replay preserves all assignments when this capacity is exceeded, rather than enlarging the receive buffer to fit the realized load. The distinction is this capacity-selection rule combined with lossless execution, rather than dropless execution, chunking or workspace reuse alone. The hardware experiments below compare capacity choices within the same replay kernel, rather than performance against these complete systems.

NCCL's GPU-initiated networking (GIN) API supports communication involving host-backed buffers (NVIDIA, 2026). This provides a communication mechanism for considering elastic receive buffers with GPU and CPU storage. In a design where assignments beyond the GPU capacity use CPU memory, MarCap could choose the GPU capacity to bound the probability of using that memory, provided that the receive layout matches the modeled assignment counts and the statistical assumptions hold. This would require integration with the buffer allocator and does not by itself bound transfer volume or latency. We leave this extension to future work.

J.2 Replay with bounded workspace

The replay mechanism we introduced in `megamoe_replay` changes the processing schedule without changing the routing decisions or the computations performed by each expert, using a positive capacity W fixed by the planner before invocation. Source inputs and routing records are prepared once, and the kernel retains the full assignment counts so that assignments beyond W can be processed in later rounds.

At destination rank c, we use the assignment count for each expert to compute where its rows begin and end in the logical sequence of L_c assignments, without first receiving all L_c rows into a buffer. The rank processes this sequence in consecutive intervals of at most W assignments each:

$$R_c = \left\lceil \frac{L_c}{W} \right\rceil, \qquad I_{c,r} = \big[rW, \min((r+1)W, L_c)\big), \quad 0 \le r < R_c. \tag{41}$$

In round r, logical row $\ell \in I_{c,r}$ uses workspace row $w = \ell - rW$, which lies in $[0, W)$. If an expert's assignments span more than one round, each round processes the corresponding portion using the same expert weights. Each rank processes its own rounds, and the number of rounds may differ between ranks.

[1] See the official Tutel MOELayer documentation and DeepSpeed MoE documentation, with the latter's capacity handling in `sharded_moe.py`.

[2] vLLM v0.10.2 fused-expert implementation, function `fused_experts_impl`.

[3] SGLang FP8 CUTLASS implementation, commit `55bf3380e073`, function `cutlass_fused_experts_fp8`.

J.3 WORKSPACE REUSE AND OUTPUT ACCUMULATION

Each routed row retains its original source rank s, token index t, and top-k slot j. After expert computation, the combine operation writes the result to the fixed source-side slot (s, t, j), where it remains until the invocation ends. Because this slot is independent of the round-local workspace address, reusing the receive buffer does not overwrite results from earlier rounds.

Within each round, dispatch, expert computation and result return are pipelined. Before reusing the buffers, the rank waits for its computation and outgoing transfers to finish and resets only the readiness state for that round, leaving the routing plan, source records and returned outputs intact. No cross-rank barrier is required between rounds, but all ranks, including those with no local destination work, participate in a final synchronization to ensure that all results have arrived. The unpermute operation then applies the routing weights and combines each token's contributions once in the original top-k order.

Algorithm 2: Capacity-bounded replay on each rank c

Input: routes, local tokens, expert weights, and fixed capacity $W > 0$.
1. Prepare source inputs and routing records, and synchronize input readiness.
2. Determine expert-ordered logical assignment ranges from the full counts.
3. **For** each local interval $I_{c,r}$ in (41):
   a. Dispatch its assignments into the W-row workspace.
   b. Compute each expert's portion using its unchanged weights.
   c. Return results to their fixed source-token/top-k slots.
   d. Complete local work before resetting round state and reusing buffers.
4. Export full counts and synchronize completion across ranks.
5. Apply routing weights and combine each source token's top-k results once.

Output: final token outputs and full per-expert assignment counts.

The disjoint round intervals cover every logical assignment exactly once, and retaining its expert identity and original output slot prevents its contribution from being dropped or duplicated when buffers are reused. Insufficient capacity increases the number of rounds without changing which assignments are evaluated, so replay correctness is guaranteed independently of MarCap's capacity estimate and statistical assumptions.

`megamoe_replay` executes its rounds on the device without per-round host intervention. We also verified its compatibility with static-shape TorchAir full-graph execution at fixed capacity in a separate EP4 test on Ascend 950PR, with outputs and expert counts exactly matching the reference results.

J.4 WORKLOADS, TIMING AND CORRECTNESS

The EP4 experiment uses the quantized configuration in Section 4, with standard experts and unquantized combine. The scratch metric is the reported total temporary workspace requirement per rank rather than total peak device memory, and it includes fixed allocation overhead but excludes model weights, external input/output buffers and separate persistent communication storage.

Table 23: **Scratch allocation for EP4 packed prefill at local batch size 512.** Receive capacity falls from 16384 to 4649 assignment rows (71.62%), while total scratch falls by 44.11%.

Workspace component (MiB)	Full reservation	MarCap
Capacity-dependent buffers	106.25	30.16
Fixed allocation overhead	66.27	66.27
Total scratch	172.52	96.43

Fixed allocation overhead includes reserved workspace, library workspace and metadata. Its share of total scratch decreases as batch size grows, so the percentage of scratch saved approaches the percentage reduction in receive-buffer capacity.

We evaluate continuous decode, Poisson prefill and decode at offered loads .75 and .95, and packed prefill at three local batch caps. For packed prefill, we use three different contiguous parts of the

held-out prefill token stream at each cap. From each part, we form 1000 measured batches in the original token order for packed contiguous prefill and another 1000 after shuffling all its tokens for packed shuffled prefill.

For each policy and scenario, we run five process repetitions, each with 16 warmup calls to avoid startup related latency followed by 1000 measured calls, and measure a kernel call's latency as the maximum rank-local event duration. We compute the overflow indicator $\mathbf{1}_{\{\max_c L_c > W\}}$ and the single-round fractions in Figure 4 from the recorded loads and configured capacity.

For the constructed inputs and weights, we compute the routing-weighted sum of fixed expert codes in FP32, convert it to BF16 and repeat it across output channels to form the reference outputs. We assign expert codes by indexing the sequence $(1, 2, 4, 8, 0.5, 0.25, 0.125, 0.0625)$ with the expert ID modulo eight, and use top-k weights $(64, 32, 16, 8, 4, 2, 1, 1)/128$. We compare the outputs of all four capacity policies with these reference outputs using `torch.equal`, and check local expert counts separately. Replay processes every assignment and safely reuses its workspace provided that routing is valid, each round computes its assigned expert outputs correctly, each assignment has a unique permanent output slot, and the round completes before its buffer is reused.

J.5 Paired comparison against full reservation

We compare full reservation, fixed capacity factors of 1 and 1.25, and MarCap using the same kernel, with MarCap capacities computed by the equivalent LP method before timing. We pair each policy with full reservation within the same scenario and process repetition before aggregating results. For mean, p50, p95 or p99 latency ℓ, the paired percentage change is $100\left(\frac{\ell_{\text{policy}}}{\ell_{\text{full}}} - 1\right)$.

For each of the nine decode configurations, we take the median paired percentage change in mean latency across repetitions, and the median of the absolute values of these nine changes is 0.36%.

Table 24: MarCap versus full reservation: paired mean-latency change (%), center [range], scratch saved (%), single-round probability (%) and mean rounds. Mean latency is computed over the measured calls in each process repetition before forming paired percentage changes. Ordinary workloads show the median and range of five paired changes, while packed workloads show the median and range of three panel medians, each formed from five repetitions. Ranges are descriptive rather than confidence intervals, and scratch savings use the reported temporary workspace requirement per rank. Single-round probability is the fraction of calls that fit in one round, and mean rounds averages the maximum round count across ranks over calls. Both columns use the same median aggregation across repetitions and panels.

Workload	Local cap	Mean latency change (%) center [range]	Scratch saved (%)	Single round (%)	Mean rounds
Continuous decode	512	-1.04 [-2.42, +4.51]	44.10	100.00	1.000
Continuous decode	2048	+0.22 [-1.55, +2.36]	63.40	100.00	1.000
Continuous decode	8192	+0.08 [-0.14, +0.25]	71.37	100.00	1.000
Poisson decode $\rho = .75$	512	+3.32 [-3.31, +5.85]	44.11	100.00	1.000
Poisson decode $\rho = .75$	2048	+0.36 [-0.95, +2.74]	63.43	100.00	1.000
Poisson decode $\rho = .75$	8192	-0.26 [-0.95, +0.76]	71.38	100.00	1.000
Poisson decode $\rho = .95$	512	-1.00 [-1.99, +4.97]	44.11	100.00	1.000
Poisson decode $\rho = .95$	2048	+0.86 [-0.10, +1.48]	63.41	100.00	1.000
Poisson decode $\rho = .95$	8192	-0.33 [-0.54, +0.35]	71.38	100.00	1.000
Poisson prefill $\rho = .75$	512	+5.84 [+1.35, +11.83]	44.10	49.00	1.510
Poisson prefill $\rho = .75$	2048	+2.41 [+1.47, +4.30]	63.43	58.30	1.417
Poisson prefill $\rho = .75$	8192	+0.77 [+0.58, +1.29]	71.37	73.50	1.265
Poisson prefill $\rho = .95$	512	+9.97 [+6.23, +11.53]	44.10	23.90	1.761
Poisson prefill $\rho = .95$	2048	+4.55 [+3.32, +4.89]	63.43	25.00	1.750
Poisson prefill $\rho = .95$	8192	+1.40 [+1.28, +1.87]	71.38	28.40	1.716
Packed shuffled	512	+0.39 [-0.43, +0.99]	44.11	100.00	1.000
Packed shuffled	2048	+0.36 [-0.22, +0.91]	63.43	100.00	1.000
Packed shuffled	8192	+0.28 [+0.21, +0.29]	71.38	100.00	1.000
Packed contiguous	512	+6.90 [+4.77, +6.94]	44.11	24.10	1.759
Packed contiguous	2048	+3.35 [+2.73, +3.85]	63.43	14.90	1.851
Packed contiguous	8192	+1.89 [+1.68, +2.22]	71.38	8.30	1.917

Table 25: MarCap versus full reservation: paired p50, p95 and p99 latency changes (%), center [range]. Paired changes are aggregated as in Table 24, and the ranges are descriptive rather than confidence intervals.

Workload	Local cap	p50 change (%) center [range]	p95 change (%) center [range]	p99 change (%) center [range]
Continuous decode	512	+0.51 [-2.77, +5.10]	-0.45 [-2.33, +5.65]	-2.41 [-13.08, +8.56]
Continuous decode	2048	-0.30 [-0.94, +1.32]	+1.42 [-1.81, +10.27]	+0.36 [+0.28, +7.62]
Continuous decode	8192	+0.00 [-0.15, +0.32]	+0.62 [-0.56, +3.03]	+0.07 [-1.30, +2.67]
Poisson decode $\rho = .75$	512	+1.03 [-2.33, +2.87]	+1.63 [-1.89, +3.84]	+1.54 [-5.22, +6.50]
Poisson decode $\rho = .75$	2048	-0.61 [-0.86, +1.93]	+1.19 [-2.28, +8.13]	+0.36 [-4.99, +7.35]
Poisson decode $\rho = .75$	8192	+0.30 [-0.96, +1.02]	-0.04 [-1.06, +0.40]	+0.14 [-3.89, +2.29]
Poisson decode $\rho = .95$	512	-0.24 [-3.84, +3.23]	-0.32 [-2.12, +7.15]	+0.87 [-1.04, +6.90]
Poisson decode $\rho = .95$	2048	+0.36 [+0.05, +1.09]	+1.67 [+0.52, +7.90]	+0.64 [-0.28, +2.04]
Poisson decode $\rho = .95$	8192	-0.13 [-0.35, +0.38]	-0.42 [-1.69, +0.57]	-0.83 [-2.30, +0.50]
Poisson prefill $\rho = .75$	512	+7.65 [+6.11, +10.75]	+5.85 [+4.57, +12.15]	+7.44 [+4.25, +11.67]
Poisson prefill $\rho = .75$	2048	+1.13 [+0.66, +2.24]	+5.68 [+5.11, +7.24]	+6.44 [+4.31, +8.71]
Poisson prefill $\rho = .75$	8192	-0.21 [-0.32, +0.13]	+2.67 [+2.18, +3.35]	+5.71 [+3.55, +6.68]
Poisson prefill $\rho = .95$	512	+11.04 [+7.85, +13.74]	+8.93 [+4.18, +16.52]	+9.26 [+7.10, +15.86]
Poisson prefill $\rho = .95$	2048	+4.45 [+3.84, +5.68]	+5.70 [+2.19, +7.18]	+5.86 [+3.19, +11.26]
Poisson prefill $\rho = .95$	8192	+1.56 [+1.27, +1.67]	+2.59 [+1.90, +3.36]	+1.84 [+0.23, +2.92]
Packed shuffled	512	+0.62 [+0.31, +1.37]	-0.49 [-1.31, +0.79]	-0.77 [-0.80, +0.91]
Packed shuffled	2048	-0.11 [-0.46, +0.13]	+1.40 [-0.31, +2.66]	+1.28 [+0.95, +1.31]
Packed shuffled	8192	+0.10 [+0.03, +0.20]	+0.83 [+0.24, +0.97]	+1.02 [+0.96, +1.95]
Packed contiguous	512	+6.31 [+6.02, +7.91]	+7.06 [+6.76, +10.16]	+6.30 [+5.13, +6.52]
Packed contiguous	2048	+3.68 [+3.17, +4.08]	+4.05 [+3.15, +5.54]	+5.64 [+4.75, +7.07]
Packed contiguous	8192	+1.86 [+1.82, +2.17]	+2.39 [+2.31, +2.95]	+2.87 [+2.67, +3.07]

Table 26: Fixed capacity factor 1 versus full reservation: paired mean-latency change (%), center [range], scratch saved (%), single-round probability (%) and mean rounds. Mean latency is computed over the measured calls in each process repetition before forming paired percentage changes. Ordinary workloads show the median and range of five paired changes, while packed workloads show the median and range of three panel medians, each formed from five repetitions. Ranges are descriptive rather than confidence intervals, and scratch savings use the reported temporary workspace requirement per rank. Round statistics are defined and aggregated as in Table 24.

Workload	Local cap	Mean latency change (%) center [range]	Scratch saved (%)	Single round (%)	Mean rounds
Continuous decode	512	+1.40 [+1.08, +8.63]	46.19	0.00	2.000
Continuous decode	2048	+3.54 [+1.01, +56.20]	64.88	0.00	2.000
Continuous decode	8192	+1.49 [+1.36, +2.97]	72.19	0.00	2.000
Poisson decode $\rho = .75$	512	+1.69 [-3.60, +8.37]	46.19	100.00	1.000
Poisson decode $\rho = .75$	2048	-0.50 [-1.14, +1.91]	64.88	100.00	1.000
Poisson decode $\rho = .75$	8192	-0.03 [-0.63, +0.09]	72.19	100.00	1.000
Poisson decode $\rho = .95$	512	-2.05 [-4.86, +3.39]	46.19	99.30	1.007
Poisson decode $\rho = .95$	2048	+0.27 [-0.95, +2.62]	64.88	92.70	1.073
Poisson decode $\rho = .95$	8192	+0.41 [+0.21, +0.48]	72.19	68.30	1.317
Poisson prefill $\rho = .75$	512	+7.52 [+0.95, +12.89]	46.19	29.00	1.710
Poisson prefill $\rho = .75$	2048	+2.42 [+2.21, +5.21]	64.88	51.00	1.490
Poisson prefill $\rho = .75$	8192	+0.84 [+0.71, +1.31]	72.19	69.40	1.306
Poisson prefill $\rho = .95$	512	+8.25 [+4.56, +11.12]	46.19	5.20	1.948
Poisson prefill $\rho = .95$	2048	+5.54 [+3.33, +6.30]	64.88	13.00	1.870
Poisson prefill $\rho = .95$	8192	+2.21 [+1.82, +2.43]	72.19	19.60	1.804
Packed shuffled	512	+8.04 [+7.60, +8.44]	46.19	0.00	2.000
Packed shuffled	2048	+3.89 [+3.40, +4.22]	64.88	0.00	2.000
Packed shuffled	8192	+1.42 [+1.17, +1.46]	72.19	0.00	2.000
Packed contiguous	512	+12.05 [+9.77, +12.28]	46.19	0.00	2.000
Packed contiguous	2048	+5.29 [+4.64, +5.57]	64.88	0.00	2.000
Packed contiguous	8192	+2.50 [+2.43, +2.70]	72.19	0.00	2.000

Table 27: Fixed capacity factor 1 versus full reservation: paired p50, p95 and p99 latency changes (%), center [range]. Paired changes are aggregated as in Table 24, and the ranges are descriptive rather than confidence intervals.

Workload	Local cap	p50 change (%) center [range]	p95 change (%) center [range]	p99 change (%) center [range]
Continuous decode	512	+6.98 [+3.66, +7.41]	+0.46 [-10.52, +8.83]	-1.83 [-16.66, +8.43]
Continuous decode	2048	+2.45 [+2.17, +10.92]	+0.38 [-3.21, +13.04]	+5.08 [+1.40, +207.91]
Continuous decode	8192	+1.62 [+1.39, +2.79]	+2.28 [+1.15, +4.24]	+2.60 [+0.81, +2.60]
Poisson decode $\rho = .75$	512	+0.74 [-2.67, +11.48]	+0.13 [-2.58, +6.47]	-2.38 [-4.02, +10.35]
Poisson decode $\rho = .75$	2048	-0.47 [-1.74, +1.95]	-2.29 [-4.57, +9.43]	-0.12 [-7.76, +2.42]
Poisson decode $\rho = .75$	8192	+0.29 [-0.50, +0.49]	+0.08 [-0.80, +0.52]	-0.66 [-1.52, -0.21]
Poisson decode $\rho = .95$	512	-1.41 [-8.45, +4.78]	-1.89 [-5.65, +2.39]	-1.21 [-4.01, +3.45]
Poisson decode $\rho = .95$	2048	+0.31 [-1.28, +1.50]	+3.01 [+0.69, +6.00]	-0.23 [-1.73, +7.58]
Poisson decode $\rho = .95$	8192	+0.19 [+0.06, +0.38]	+1.38 [+0.41, +1.79]	+0.03 [-2.01, +1.08]
Poisson prefill $\rho = .75$	512	+8.89 [+5.75, +12.17]	+8.73 [-2.72, +12.93]	+7.34 [-0.70, +14.85]
Poisson prefill $\rho = .75$	2048	+2.48 [+1.60, +4.98]	+6.55 [+5.15, +8.78]	+5.41 [+2.84, +10.45]
Poisson prefill $\rho = .75$	8192	-0.27 [-0.72, -0.17]	+2.57 [+2.29, +2.98]	+3.58 [+1.53, +5.16]
Poisson prefill $\rho = .95$	512	+10.42 [+5.68, +12.40]	+7.25 [+1.55, +17.02]	+6.71 [+3.83, +9.70]
Poisson prefill $\rho = .95$	2048	+5.51 [+3.90, +6.42]	+6.22 [+3.04, +7.98]	+9.13 [+2.56, +10.63]
Poisson prefill $\rho = .95$	8192	+2.13 [+2.05, +2.49]	+2.78 [+1.96, +4.12]	+3.10 [+2.26, +3.49]
Packed shuffled	512	+8.04 [+7.81, +8.81]	+6.00 [+5.64, +7.27]	+3.71 [+3.57, +9.55]
Packed shuffled	2048	+3.63 [+3.17, +3.92]	+4.33 [+2.99, +5.50]	+3.29 [+2.47, +3.50]
Packed shuffled	8192	+1.39 [+0.87, +1.58]	+1.43 [+0.46, +1.85]	+3.83 [+1.02, +4.78]
Packed contiguous	512	+11.13 [+10.81, +11.73]	+8.20 [+4.90, +8.81]	+8.35 [+5.44, +8.44]
Packed contiguous	2048	+5.04 [+5.04, +5.09]	+5.20 [+4.78, +5.96]	+5.48 [+4.03, +7.83]
Packed contiguous	8192	+2.57 [+2.55, +2.71]	+3.16 [+2.33, +3.36]	+3.04 [+2.97, +3.07]

Table 28: Fixed capacity factor 1.25 versus full reservation: paired mean-latency change (%), center [range], scratch saved (%), single-round probability (%) and mean rounds. Mean latency is computed over the measured calls in each process repetition before forming paired percentage changes. Ordinary workloads show the median and range of five paired changes, while packed workloads show the median and range of three panel medians, each formed from five repetitions. Ranges are descriptive rather than confidence intervals, and scratch savings use the reported temporary workspace requirement per rank. Round statistics are defined and aggregated as in Table 24.

Workload	Local cap	Mean latency change (%) center [range]	Scratch saved (%)	Single round (%)	Mean rounds
Continuous decode	512	-2.65 [-3.74, +0.16]	42.34	100.00	1.000
Continuous decode	2048	-0.15 [-1.54, +0.45]	59.48	100.00	1.000
Continuous decode	8192	-0.10 [-0.38, +0.16]	66.17	100.00	1.000
Poisson decode $\rho = .75$	512	-0.37 [-1.46, +0.99]	42.34	100.00	1.000
Poisson decode $\rho = .75$	2048	-0.32 [-1.89, +0.74]	59.48	100.00	1.000
Poisson decode $\rho = .75$	8192	-0.12 [-0.71, +0.28]	66.17	100.00	1.000
Poisson decode $\rho = .95$	512	-0.38 [-4.74, +1.86]	42.34	100.00	1.000
Poisson decode $\rho = .95$	2048	-0.38 [-0.55, +0.36]	59.48	100.00	1.000
Poisson decode $\rho = .95$	8192	-0.19 [-0.54, +0.00]	66.17	100.00	1.000
Poisson prefill $\rho = .75$	512	+0.94 [-0.68, +5.24]	42.34	76.10	1.239
Poisson prefill $\rho = .75$	2048	+0.59 [-0.63, +2.64]	59.48	90.80	1.092
Poisson prefill $\rho = .75$	8192	-0.18 [-0.55, +0.12]	66.17	99.10	1.009
Poisson prefill $\rho = .95$	512	+3.96 [+0.41, +4.99]	42.34	62.50	1.375
Poisson prefill $\rho = .95$	2048	+1.10 [+0.61, +1.64]	59.48	86.40	1.136
Poisson prefill $\rho = .95$	8192	+0.20 [-0.13, +0.26]	66.17	97.40	1.026
Packed shuffled	512	+0.50 [-0.55, +1.21]	42.34	100.00	1.000
Packed shuffled	2048	-0.34 [-0.77, -0.10]	59.48	100.00	1.000
Packed shuffled	8192	+0.07 [-0.05, +0.08]	66.17	100.00	1.000
Packed contiguous	512	+1.97 [+1.25, +2.58]	42.34	64.30	1.357
Packed contiguous	2048	+0.28 [-0.02, +0.32]	59.48	89.80	1.102
Packed contiguous	8192	-0.00 [-0.03, +0.27]	66.17	98.30	1.017

Table 29: Fixed capacity factor 1.25 versus full reservation: paired p50, p95 and p99 latency changes (%), center [range]. Paired changes are aggregated as in Table 24, and the ranges are descriptive rather than confidence intervals.

Workload	Local cap	p50 change (%) center [range]	p95 change (%) center [range]	p99 change (%) center [range]
Continuous decode	512	-0.80 [-1.08, +0.34]	+0.59 [-5.40, +4.02]	-1.41 [-17.87, +3.29]
Continuous decode	2048	-0.24 [-1.13, +0.48]	+0.96 [-2.81, +1.32]	+0.54 [-3.09, +2.94]
Continuous decode	8192	-0.05 [-0.33, +0.05]	-0.06 [-0.76, +2.09]	+0.02 [-1.23, +1.26]
Poisson decode $\rho = .75$	512	-0.04 [-1.35, +0.96]	-1.50 [-2.07, -0.49]	-1.25 [-2.83, +0.39]
Poisson decode $\rho = .75$	2048	-0.47 [-1.55, -0.10]	-0.61 [-1.69, +1.76]	+0.19 [-3.30, +1.68]
Poisson decode $\rho = .75$	8192	-0.05 [-0.96, +0.22]	-0.08 [-0.80, +0.61]	+0.18 [-5.71, +7.66]
Poisson decode $\rho = .95$	512	-1.82 [-6.05, +0.15]	-1.80 [-3.97, +1.93]	+0.84 [-5.97, +6.94]
Poisson decode $\rho = .95$	2048	-0.35 [-1.38, +0.38]	-0.23 [-0.72, +4.79]	+0.17 [-0.53, +2.33]
Poisson decode $\rho = .95$	8192	-0.10 [-0.35, +0.14]	-0.88 [-1.16, +0.37]	-1.38 [-1.91, -0.38]
Poisson prefill $\rho = .75$	512	+2.18 [-0.68, +3.51]	+2.68 [+0.37, +7.47]	+4.39 [+1.42, +6.78]
Poisson prefill $\rho = .75$	2048	-0.35 [-2.14, +2.61]	+2.93 [+1.88, +3.27]	+3.48 [-0.10, +12.60]
Poisson prefill $\rho = .75$	8192	-0.32 [-0.69, +0.18]	-0.30 [-0.58, +0.58]	+1.56 [-1.95, +2.16]
Poisson prefill $\rho = .95$	512	+4.21 [+0.58, +4.50]	+3.95 [-2.04, +10.17]	+6.22 [+5.53, +20.96]
Poisson prefill $\rho = .95$	2048	+0.42 [-0.15, +0.98]	+2.71 [+0.29, +3.45]	+3.67 [+1.86, +5.19]
Poisson prefill $\rho = .95$	8192	-0.02 [-0.19, +0.12]	+0.34 [+0.13, +1.32]	+0.04 [-0.43, +2.45]
Packed shuffled	512	+2.22 [+0.11, +2.44]	+0.18 [-0.79, +0.65]	-1.03 [-3.86, +2.02]
Packed shuffled	2048	+0.03 [-0.52, +0.34]	-0.50 [-1.69, -0.39]	-0.29 [-1.12, +0.01]
Packed shuffled	8192	+0.05 [-0.05, +0.06]	-0.10 [-0.23, +0.03]	+0.55 [+0.37, +0.96]
Packed contiguous	512	+2.15 [+1.86, +2.31]	+1.64 [+1.42, +4.20]	+1.54 [+0.44, +3.82]
Packed contiguous	2048	-0.08 [-0.19, +0.31]	+1.41 [+1.30, +1.72]	+3.55 [-0.07, +5.37]
Packed contiguous	8192	+0.01 [-0.07, +0.16]	+0.54 [+0.00, +0.68]	+1.13 [+0.42, +1.76]

J.6 Latency mechanisms

Packed contiguous and shuffled workloads use the same tokens, aggregate marginals, capacity and placement, but their latencies differ even under full reservation (Table 30). Because full reservation completes every batch in one round, these differences are not caused by additional replay rounds. Therefore, token order can affect latency through other factors, possibly including changes in load balance and straggling.

Table 30: Full reservation: paired reduction in mean latency (%) from shuffling packed prefill tokens, relative to contiguous order.

Local batch cap	Mean latency reduction (%)
512	5.01
2048	13.21
8192	13.55

A separate balanced-route control measures the overhead of extra rounds by forcing one, two or four rounds rather than letting MarCap choose the capacity. Table 31 reports p50 overheads from two process measurements per setting, each with 50 timing samples.

Table 31: p50 latency overhead of forced replay rounds relative to one round, in percent [range across two process measurements].

Local batch	Two rounds	Four rounds
1024	7.12 [6.93, 7.32]	19.06 [18.44, 19.68]
4096	0.61 [0.52, 0.70]	6.25 [6.18, 6.31]
8192	0.23 [0.19, 0.27]	0.70 [0.70, 0.70]