DyS-MoE: Virtualizing Dynamic Expert Execution with Static Windows for Resource-Constrained MoE Inference

Anonymous authors
Paper under double-blind review

Abstract

Mixture-of-Experts (MoE) inference under limited GPU memory relies on expert offloading. Dynamic execution exploits heterogeneous resources but incurs frequent transfers and scheduling overhead, while static expert placement simplifies execution at the cost of GPU utilization. We present **DyS-MoE**, which uses selective expert replacement to serve dynamic demand through reusable execution windows. Its core abstraction, a *Virtualized Static Execution Graph*, preserves operations and dependencies while expert bindings change. *Slotted Grouped Execution* provides stable GPU cache and execution slots, and *Importance-Aware Miss Handling* maps residency misses and capacity overflow onto these resources while reserving CPU exact computation for sensitive contributions. Together, these mechanisms enable CUDA Graph replay under dynamic expert offloading. On an RTX 4090 at batch 8, DyS-MoE achieves 431.99 and 551.06 decode tokens/s on Qwen3-30B-A3B and DeepSeek-V2-Lite-Chat, respectively, reaching 2.28× and 6.90× the throughput of state-of-the-art baselines while maintaining competitive task accuracy.

1 Introduction

Mixture-of-Experts (MoE) models expand parameter capacity while activating a small subset of experts for each token (Shazeer et al., 2017; Fedus et al., 2022; Jiang et al., 2024; DeepSeek-AI, 2024; Qwen Team, 2025). This sparsity makes large models attractive for local inference, but their weights can exceed the memory of a consumer GPU. Offloaded inference stores the full expert set in host memory and keeps a working set on the GPU (Eliseev & Mazur, 2023; Xue et al., 2025; Du et al., 2024). Each decode step must then turn changing expert requests into useful computation across the two devices.

Dynamic expert scheduling coordinates CPU computation, GPU execution, caching, and prefetch to exploit heterogeneous resources (Xue et al., 2025; Kamahori et al., 2025; Zhong et al., 2025). However, GPU misses still depend on weight transfers or CPU results, and changing execution paths require repeated launches and coordination. Fixed CPU-expert placement offers a regular path but can leave the GPU waiting for CPU computation (Chen et al., 2025). The challenge is to serve changing expert requests through a reusable execution structure without making the CPU the dominant compute path.

Our insight is to *decouple dynamic expert demand from execution structure*. Expert redundancy permits selected requests to use similar, available experts (Wang et al., 2025; Wu et al., 2026). A miss can then change its expert binding while preserving the operation sequence. Bounded CPU exact computation protects contributions that are costly to approximate.

We present **DyS-MoE**, jointly designing execution windows, expert slots, and miss handling to make dynamic demand compatible with reusable execution. The key is to use replacement to absorb residency misses and capacity overflow within prepared resources, allowing dynamic expert offloading to retain a replayable execution structure. Our contributions are:

- **Virtualized Static Execution Graph.** We introduce an execution abstraction that expresses changing expert demand through updated bindings within reusable windows. The

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.

windows span adjacent layers to retain GPU routing and binding in the captured sequence, enabling CUDA Graph replay alongside bounded CPU computation and prefetch.

- **Slotted Grouped Execution.** We expose the GPU expert cache through stable slots, separating expert weight storage from assignment capacity. Exact and replacement assignments share one grouped execution path, so changing expert availability does not require rebuilding that path; capacity grouping further reduces long-tail padding.
- **Importance-Aware Miss Handling.** We combine GPU replacement with bounded CPU exact computation in a common procedure for residency misses and capacity overflow. Routing importance and calibrated replacement error guide exact-computation priority, while residual requests use available GPU capacity without changing the execution structure.
- **System evaluation.** On an RTX 4090, DyS-MoE achieves 2.28× and 6.90× the throughput of the fastest evaluated baseline configurations on Qwen3-30B-A3B and DeepSeek-V2-Lite-Chat at batch size 8. Controlled ablations identify the sources of speedup and the effects of miss handling on task quality.

2 Background and Related Work

2.1 MoE Workloads and Expert Computation

A router selects experts for each token and combines their outputs using routing weights (Shazeer et al., 2017; Fedus et al., 2022). We call each token–expert computation an *assignment*. Routing changes both expert identities and per-expert loads. Systems such as MegaBlocks, Tutel, and FasterMoE organize sparse expert computation and address load imbalance (Gale et al., 2023; Hwang et al., 2023; He et al., 2022). When weights exceed GPU memory, execution must additionally account for expert availability (Eliseev & Mazur, 2023).

2.2 Offloading and Execution Paths

Dynamic expert scheduling adapts computation and transfers to runtime requests. MoE-Infinity uses activation traces for caching and prefetch; SiDA, Pre-gated MoE, and FATE predict future expert demand (Xue et al., 2025; Du et al., 2024; Hwang et al., 2024; Fang et al., 2025). Fiddler selects between CPU computation and GPU transfer, HybriMoE combines intra-layer scheduling with caching and prefetch, and LayerScope coordinates cross-layer loads (Kamahori et al., 2025; Zhong et al., 2025; Yu et al., 2026). These approaches improve resource overlap, but GPU misses still depend on transfers or CPU results.

Static expert placement, illustrated by layer offloading in llama.cpp and the CPU-expert KTransformers configuration, reduces placement decisions (Gerganov, 2023; Chen et al., 2025). CUDA Graph separately amortizes submission overhead through replay (NVIDIA, 2019). Placement determines where computation resides; replay requires reusable operations and dependencies. DyS-MoE retains the latter while allowing expert bindings to change.

2.3 Expert Redundancy and Replacement

MC-SMoE exploits redundancy by merging and compressing experts (Li et al., 2024). Runtime replacement instead retains the expert pool and changes which experts serve selected requests. BuddyMoE uses co-activation and output similarity to replace prefetch misses; SERE reroutes secondary experts to similar primary experts while retaining critical experts (Wang et al., 2025; Wu et al., 2026). DyS-MoE builds on runtime replacement to address execution reuse: residency misses and capacity overflow are mapped to prepared slots so that expert selection can change without changing the captured operation sequence. Its contribution is the joint design of replacement, slot capacity, and execution windows for CUDA Graph replay under dynamic expert offloading, with bounded CPU exact computation protecting sensitive requests.

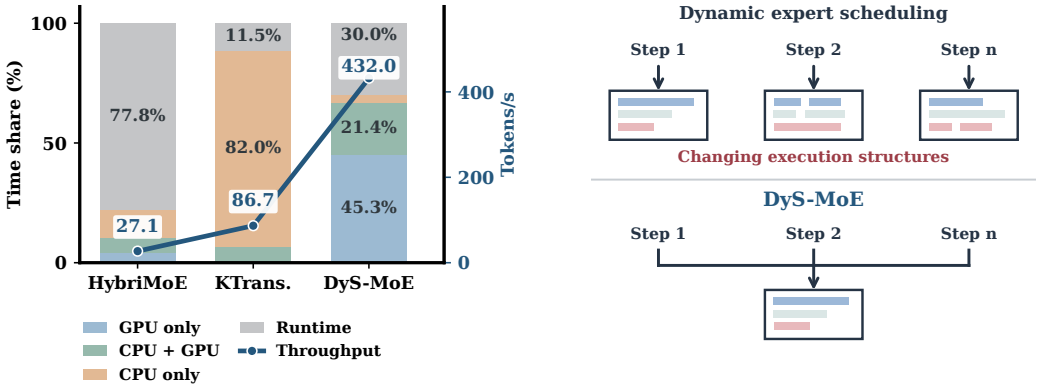
3 MOTIVATION

Serving dynamic expert requests efficiently requires reducing execution overhead, organizing reusable GPU resources, and handling requests that do not fit their original expert slots. We derive these requirements in turn.

3.1 DYNAMIC REQUESTS AND STABLE EXECUTION

The Qwen time breakdown at batch size 8 in Figure 1(a) shows two different costs: CPU-only execution accounts for 82.0% of KTransformers' profiled time, while runtime accounts for 77.8% of HybriMoE's. DyS-MoE instead spends 66.7% in GPU computation, including CPU-overlapped intervals. The two baselines expose complementary obstacles: reducing submission overhead does not shorten CPU-dominated computation, while moving work to the GPU still leaves execution-organization costs. Dynamic expert scheduling creates many short operations and cross-device dependencies. A Qwen trace at batch size 8 of HybriMoE's calibrated batched configuration records approximately 9,738 kernels per decode step. Replaying these operations can amortize submission, but cannot by itself remove dependencies on missing expert weights or CPU results. Figure 2 illustrates these execution paths; the replay ablation in Section 5.3 measures the benefit of amortizing submission in DyS-MoE.

Routing can vary assignment indices within a fixed graph. Offloading adds the question of where each expert executes and when its output becomes available. If every miss changes the compute or transfer path, capturing the resident GPU path alone does not resolve this uncertainty. Figure 1(b) illustrates our alternative: change request bindings within a reusable structure. This motivates the *Virtualized Static Execution Graph*: expert requests may change, while the operations and dependencies serving them remain reusable.



(a) Placement and execution cost **(b) Execution window reuse**

Figure 1: Execution time and reusable structure. (a) Qwen profiled time shares at batch size 8 (bars) and independently measured decode throughput (line). Green denotes CPU–GPU overlap, counted once; blue and orange denote GPU-only and CPU-only time. Settings and normalization are given in Appendix C.3. (b) Dynamic expert scheduling changes execution structure across decode steps; DyS-MoE serves changing expert requests through reusable execution windows. Operation blocks are schematic.

3.2 EXPERT SLOTS AND GPU WORK ORGANIZATION

Stable weight addresses alone do not provide a reusable execution layout: changing expert demand also changes how assignments occupy computation resources. Even with unchanged resident weights, one expert may receive many assignments in one step and few in the next. A reusable structure must therefore accommodate changing loads as well as changing expert identities. Persistent cache and execution slots separate storage and capacity from their occupants: bindings specify which experts and assignments use these resources without rebuilding the execution layout.

Long-tailed loads make a common maximum capacity wasteful, while smaller capacities can overflow despite resident weights. *Slotted Grouped Execution* provides the reusable slot layout, with capacity grouping reducing reservation for the tail. The remaining requirement is to serve requests whose original experts are unavailable or whose slots are full.

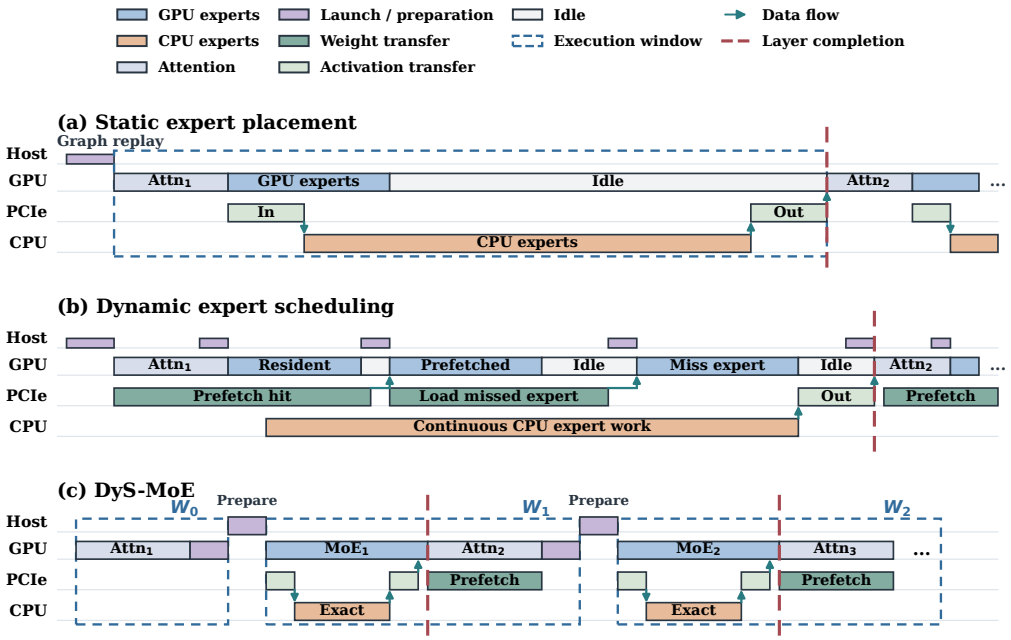


Figure 2: Execution patterns of expert offloading. (a) Static expert placement can leave the GPU idle when CPU expert computation dominates. (b) Dynamic expert scheduling interleaves computation with kernel launches and weight transfers. (c) DyS-MoE overlaps GPU experts, bounded CPU computation, and prefetch within reusable windows spanning MoE_i and Attention_{i+1}, with brief host preparation between windows. Dashed boxes mark execution windows, arrows indicate data flow, and red dashed lines mark layer completion.

3.3 Miss Handling, Execution Stability, and Quality

Residency misses and capacity overflow must both be served within the prepared resources for the execution structure to remain reusable. Loading every missed expert introduces transfer dependencies; computing every miss on the CPU can make CPU execution the bottleneck. Prior work uses expert replacement to serve selected requests with available alternatives (Wang et al., 2025; Wu et al., 2026). DyS-MoE applies this principle to both residency misses and execution-slot capacity overflow.

Replacement quality varies across experts and inputs. Two misses can have very different effects on the layer output: one has a close available substitute, while another has a large routing weight and no good alternative. Treating them equally spends scarce CPU work without considering the output it protects. Routing importance and calibrated replacement error together provide a basis for choosing which requests receive exact computation. *Importance-Aware Miss Handling* therefore reserves bounded CPU work for sensitive requests and assigns the remainder to available GPU capacity. Together, windows fix the operation sequence, slots stabilize resources, and miss handling maps changing demand to those resources.

4 DyS-MoE

DyS-MoE turns changes in routed experts and per-expert demand into binding updates over fixed execution resources (Figure 3). Windows define the operation sequence, slots expose its resources, and

miss handling assigns requests to GPU or bounded CPU computation. Offline preparation supplies calibration and execution budgets.

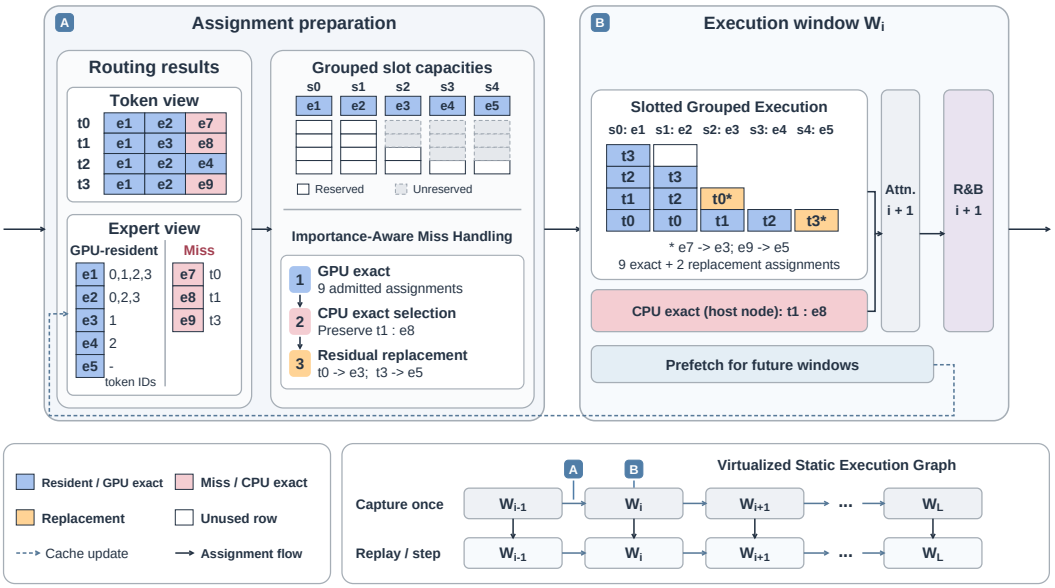


Figure 3: DyS-MoE overview. (A) Assignment preparation maps routed requests to fixed-capacity slots through GPU exact admission, CPU exact selection, and residual replacement. (B) Execution window W_i overlaps GPU expert execution, CPU exact computation, and prefetch, followed by next-layer attention and routing and binding (R&B). Matching A/B markers locate the supplied bindings and window execution in the replay sequence. Symbols t, e, and s denote tokens, experts, and slots; stars mark replacements. White rows are reserved but unused, gray dashed rows are unreserved, and the dashed feedback arrow denotes cache updates.

4.1 Virtualized Static Execution Graph

The Virtualized Static Execution Graph fixes operations, buffers, and dependencies while treating expert identities and token assignments as mutable bindings. Each layer-specific window consumes the bindings for layer i and produces those for layer $i+1$. Its main computation sequence is

$$W_i = [\mathrm{MoE}_i,\ \mathrm{Attention}_{i+1},\ \mathrm{Router}_{i+1},\ \mathrm{Bind}_{i+1}]. \tag{1}$$

The window organizes this captured GPU sequence together with overlapping CPU exact computation and prefetch. Its boundary therefore describes the execution organization, not the capture scope of every concurrent operation. CPU and GPU expert outputs merge before attention; optional prefetch supplies future bindings. Window replay does not require prefetch. The shifted boundary keeps next-layer GPU routing and binding inside the captured sequence, avoiding separate host submissions for these operations and preparing the next MoE without rebuilding its execution path. Host preparation between windows handles work that cannot be captured (Figure 2). Stable buffers and dependencies allow each layer's window to replay across decode steps with updated bindings. Execution constraints are detailed in Appendix A.3.

4.2 Slotted Grouped Execution

Slot organization separates weight storage from execution capacity. Cache slots hold expert weights; execution slots provide fixed assignment capacities and persistent buffers. Binding maps available experts and token assignments onto these resources. Slots remain fixed while their bindings change, allowing expert availability to vary without rebuilding the graph. Resident and prefetched experts supply both exact and replacement targets, which share the same grouped execution path. Slot selection and operator details are given in Appendix A.2.

Capacity grouping further improves this layout for long-tailed loads. Larger slots accommodate hot experts and smaller slots cover the tail, reducing reservation and padding relative to a common maximum width (Appendix A.2, Figure 5). For slot s with capacity c_s,

$$n_s^{\text{exact}} + n_s^{\text{repl}} \le c_s, \qquad C = \sum_s c_s. \tag{2}$$

Requests exceeding their original slot's fixed capacity enter Importance-Aware Miss Handling, so overflow changes bindings rather than the graph. Appendix B.3 gives the capacity condition ensuring that every assignment can be served.

4.3 Importance-Aware Miss Handling

Importance-Aware Miss Handling allocates requests in three stages: GPU exact admission, bounded CPU exact selection, and residual GPU replacement (Figure 3A). This order uses available exact computation first, protects high-impact misses, and fits the remaining requests to the fixed GPU execution resources. Let $a = (t_a, e_a, r_a, w_a)$ denote an assignment with token t_a, requested expert e_a, zero-based router rank r_a, and weight w_a. GPU exact execution requires both available weights and a row in the requested expert's slot. The dispatcher first admits these requests, giving higher-importance assignments priority when an expert's demand exceeds its capacity. Assignments without an available expert slot or sufficient capacity form the unresolved set $\mathcal{M}$ and enter the same CPU selection and GPU replacement procedure.

Protecting high-impact requests. For each layer-step, a shared CPU exact budget protects unresolved assignments with high routing importance and poor substitutes. We use the heuristic scores

$$I(a) = \frac{w_a}{r_a + 1}, \qquad S(a) = I(a)\, E_\ell^{\text{cpu}}(e_a; \mathcal{V}). \tag{3}$$

Here $\mathcal{V}$ contains all currently available resident and ready-prefetched experts. $E_\ell^{\text{cpu}}(e_a; \mathcal{V})$ uses the best calibrated substitute with spare capacity after initial routing admission, before any replacement (Appendix B.1). The rank factor gives additional priority to earlier router choices, while w_a distinguishes contribution magnitudes among assignments at the same rank. This heuristic emphasizes leading contributions when capacity is scarce. Thus, $I(a)$ resolves competition for GPU capacity, while $S(a)$ favors contributions that are both important and difficult to approximate. The highest-scoring eligible requests receive CPU exact computation, up to B_{cpu} assignments per layer-step.

Binding residual replacements. Residual requests traverse candidates in calibrated error order. At each candidate rank, requests competing for the same target are admitted by $I(a)$ within its remaining capacity; rejected requests try the next candidate. CPU-selected requests are excluded, and accepted replacements share grouped execution with GPU exact assignments. Algorithm 1 in Appendix A.1 gives the complete procedure.

Combining exact and replacement outputs. For token t, let $\mathcal{X}_t$ denote assignments executed on their requested experts and $\mathcal{R}_t$ those bound to replacement experts. The routed-expert output is

$$\hat{y}_t = \sum_{a \in \mathcal{X}_t} w_a f_{e_a}(h_t) + \sum_{a \in \mathcal{R}_t} w_a f_{\tilde{e}_a}(h_t), \tag{4}$$

Here h_t is the input state and f_e is expert e. Exact assignments run on either device. Replacement uses $\tilde{e}_a$ while preserving the original router weight w_a; the priority score does not reweight the output.

4.4 Offline Preparation

Offline preparation combines model calibration on representative inputs with execution profiling on the target hardware.

Model calibration. Calibration traces supply resident-expert profiles, replacement estimates, CPU-priority statistics, and prefetch-predictor training data. For source expert e at layer ℓ, let $\mathcal{H}_{\ell,e}$

contain inputs sampled from exact-model traces where e was routed. We compare source and candidate outputs on the same inputs:

$$D_\ell(e,j) = \sqrt{\frac{1}{|\mathcal{H}_{\ell,e}|}\sum_{h\in\mathcal{H}_{\ell,e}}\left(\frac{\mathrm{RMS}(f_{\ell,e}(h)-f_{\ell,j}(h))}{\max(\mathrm{RMS}(f_{\ell,e}(h)),\epsilon)}\right)^2}, \tag{5}$$

This normalized discrepancy ranks GPU substitutes; the best calibrated alternative among currently available experts informs CPU priority. ϵ stabilizes the denominator.

Routed-demand profiles select the initial resident experts, and a calibrated cross-layer predictor ranks future experts for prefetch from current hidden states. Resident and prefetched experts jointly supply exact hits and replacement candidates. The target-layer router still determines the requested experts and their weights. Predictor structure and measured overhead are detailed in Appendix B.2.

Hardware-aware budget selection. Profiling the target model, hardware, batch size, and cache configuration identifies CPU and prefetch budgets whose work can overlap GPU execution. Within that range, larger budgets offer more exact-execution opportunities; prefetch degree zero is also admissible. This criterion guides execution budgeting rather than optimizing downstream accuracy, which is evaluated separately. Appendix B.4 details the procedure and offline cost.

We implement DyS-MoE in KTransformers using native dispatch and grouped GPU operators. Implementation details are given in Appendix A.

5 Evaluation

Our evaluation first measures whether reusable execution delivers higher throughput while preserving task quality under limited GPU memory. Component ablations then examine the roles of window replay, capacity grouping, and exact computation. Finally, budget sweeps characterize how much CPU work and prefetch the execution pipeline can accommodate.

5.1 Experimental Setup

We evaluate DeepSeek-V2-Lite-Chat and Qwen3-30B-A3B (DeepSeek-AI, 2024; Qwen Team, 2025) on an RTX 4090 (24 GB), an Intel Core i9-14900K, and 64 GB host memory, using Q4_K_M weights and BF16 GPU expert computation. Unless varied, DyS-MoE uses batch size 8, a 40% nominal expert cache ratio, one CPU exact assignment per layer-step, prefetch degree 2, and capacity tiers $[8, 4, 2]$. CPU budget 1 and prefetch degree 2 define a common comparison point across tasks, rather than a per-task optimum.

Workloads and metrics. We report decoding throughput in tokens per second. Quality evaluation reports GSM8K accuracy (Cobbe et al., 2021) and HumanEval pass@1 (Chen et al., 2021) with greedy decoding. GSM8K uses four-shot prompts and a 1,024-token generation limit; HumanEval uses chat-formatted function completion and a 512-token limit. HumanEval calibration uses MBPP+ calibration prompts (Austin et al., 2021; Liu et al., 2023) (Appendix B). We additionally evaluate calibration transfer on StrategyQA (Geva et al., 2021) (Appendix D.6).

Baselines. We compare with llama.cpp, KTransformers, and HybriMoE (Gerganov, 2023; Chen et al., 2025; Zhong et al., 2025), covering static expert placement and dynamic expert scheduling. We compare throughput across batch sizes and expert cache ratios. Baseline configurations and evaluation protocols appear in Appendix C; GPU memory footprints are reported in Appendix D.5.

5.2 End-to-End Performance

DyS-MoE sustains its throughput advantage across batch sizes (Figure 4). At batch size 16, DyS-MoE reaches 617.46 and 1066.98 tokens/s on Qwen and DeepSeek, respectively: 4.24× and 7.20× the fastest baseline in the batch sweep. Throughput grows by 7.90× and 14.56× from batch size 1 to 16; Appendix D.1 provides the plotted values.

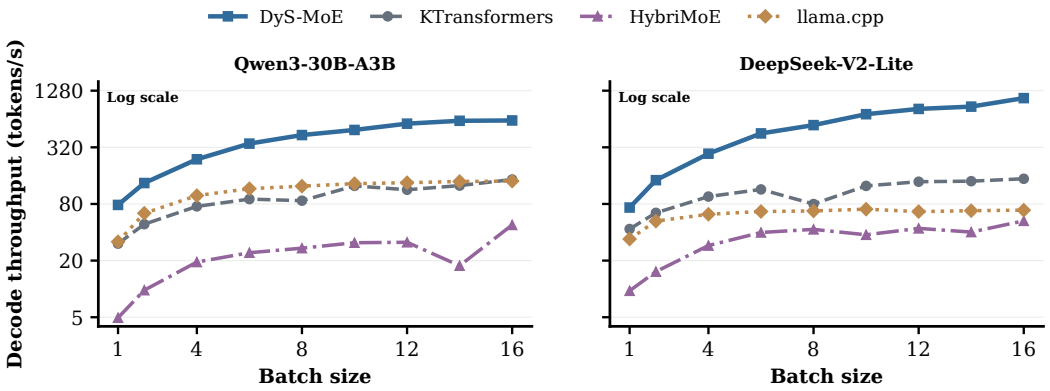


Figure 4: Decode throughput versus batch size on a logarithmic scale. DyS-MoE and HybriMoE use 40% expert residency; DyS-MoE uses CPU budget 1 and prefetch degree 2. Baseline configurations are described in Appendix C.

Table 1: Throughput and task quality across cache ratios at batch size 8. Parentheses give accuracy changes from the original model in percentage points; bold denotes the highest throughput. KT denotes KTransformers, whose CPU-expert configuration is unchanged across ratios.

	Decode throughput (tokens/s)				DyS-MoE accuracy (%)	
Cache	KT	HybriMoE	llama.cpp	DyS-MoE	GSM8K	HumanEval
Qwen3-30B-A3B (original accuracy)					92.34	89.63
20%	86.68	25.24	153.29	**484.52**	89.54 (−2.80)	87.20 (−2.43)
40%	86.68	27.11	189.07	**431.99**	92.95 (+0.61)	87.20 (−2.43)
60%	86.68	28.43	257.09	**364.15**	92.65 (+0.31)	89.02 (−0.61)
DeepSeek-V2-Lite (original accuracy)					59.06	34.76
20%	79.88	50.67	74.05	**553.31**	42.68 (−16.38)	20.12 (−14.64)
40%	79.88	39.54	78.94	**551.06**	55.57 (−3.49)	32.93 (−1.83)
60%	79.88	44.51	87.64	**513.04**	57.85 (−1.21)	37.20 (+2.44)

Performance under limited residency. At the default 40% cache ratio, DyS-MoE delivers 431.99 and 551.06 tokens/s on Qwen and DeepSeek, respectively, exceeding the strongest baseline in Table 1 by 2.28× and 6.90×. Qwen retains the original model's GSM8K accuracy with a 2.43-point decrease on HumanEval; DeepSeek remains within 3.49 points on both tasks. At this operating point, the execution structure supports substantial throughput gains while selective replacement and bounded exact computation largely preserve task quality.

Calibration transfer. Without StrategyQA calibration examples, Qwen calibrated on GSM8K and MBPP+ achieves 65.50% on StrategyQA, versus 69.00% for the exact reference. Appendix D.6 reports the source-calibration comparisons.

Effect of cache ratio. DyS-MoE maintains the highest throughput at every evaluated cache ratio (Table 1). Increasing residency from 20% to 60% improves DeepSeek's GSM8K and HumanEval scores by 15.16 and 17.08 points, respectively, with a 7.3% throughput reduction. Qwen is less sensitive in quality, while its throughput decreases by 24.8%. Greater residency offers more opportunities for exact expert execution, but may also spread assignments across more experts and increase weight-access costs. The results therefore favor choosing cache capacity jointly with quality and throughput, rather than maximizing residency alone.

5.3 Component Analysis

Table 2 separates the role of execution reuse from the mechanisms that preserve task quality. Each ablation removes one component, with Full DyS-MoE as the reference.

Table 2: Component ablations at batch size 8. Throughput ratios and parenthesized accuracy changes (percentage points) are relative to Full DyS-MoE. Graph-off quality scores are inherited from Full, with expert selection and computation unchanged.

Configuration	Tokens/s	Relative	GSM8K (%)	HumanEval (%)
Qwen3-30B-A3B				
Full DyS-MoE	441.58	1.00×	92.95	87.20
w/o CUDA Graph	159.99	0.36×	92.95 (+0.00)	87.20 (+0.00)
w/o capacity grouping	423.80	0.96×	92.65 (−0.30)	87.20 (+0.00)
w/o CPU compute	475.50	1.08×	91.89 (−1.06)	84.76 (−2.44)
w/o prefetch	446.91	1.01×	93.03 (+0.08)	88.41 (+1.22)
DeepSeek-V2-Lite				
Full DyS-MoE	567.15	1.00×	55.57	32.93
w/o CUDA Graph	139.75	0.25×	55.57 (+0.00)	32.93 (+0.00)
w/o capacity grouping	537.04	0.95×	54.66 (−0.91)	31.71 (−1.22)
w/o CPU compute	643.17	1.13×	50.72 (−4.85)	21.95 (−10.98)
w/o prefetch	825.82	1.46×	54.89 (−0.68)	34.15 (+1.22)

Execution reuse. Window replay improves throughput by 2.76× on Qwen and 4.06× on DeepSeek with the other components unchanged, establishing the benefit of the reusable execution path. Capacity grouping adds 4.19% and 5.61% throughput, respectively, by refining the slot layout for long-tailed loads.

Exact computation and prefetch. Bounded CPU exact computation preserves sensitive assignments: removing it reduces HumanEval accuracy by 2.44 points on Qwen and 10.98 points on DeepSeek, for throughput gains of 7.68% and 13.40%, respectively. Prefetch is an optional cache-update mechanism: it improves DeepSeek GSM8K accuracy by 0.68 points, but disabling it raises throughput by 46% and HumanEval accuracy by 1.22 points. Qwen also retains comparable quality without prefetch. Prefetch can supply original experts and additional replacement candidates, but transfer costs and changed replacement choices make its value workload-dependent; it is not required for our replay mechanism.

Importance-aware miss handling. With unchanged CPU and prefetch budgets, random CPU selection and GPU replacement reduce Qwen GSM8K accuracy from 92.95% to 90.07% and HumanEval from 87.20% to 76.22%. Porting BuddyMoE and SERE offline procedures and replacement-priority policies into our runtime yields 84.15% and 86.59% on HumanEval, respectively, versus 87.20% for DyS-MoE (Appendix D.3).

Execution budgets. The Qwen sweeps (Appendix D.4) show diminishing returns from CPU work and non-monotonic prefetch benefits, supporting budget selection within the available execution overlap.

6 Conclusion

DyS-MoE uses selective expert replacement to make dynamic MoE requests compatible with reusable execution. Windows preserve the operation sequence, slots support changing bindings, and importance-aware miss handling protects sensitive contributions with bounded exact computation. At batch size 8 on an RTX 4090, DyS-MoE achieves 2.28× and 6.90× the throughput of the strongest compared baselines on Qwen and DeepSeek, with competitive task quality. Flexibility in expert selection enables execution reuse without requiring every expert to reside on the GPU.

AI USE STATEMENT

The authors conceived the research ideas, developed the methodological design, and directed the implementation and evaluation. Generative AI tools were used under author guidance to assist with coding, experimental workflows, manuscript preparation, figure-generation code, and supporting analyses. The authors made the final methodological and interpretive decisions and take responsibility for the paper's content and results.

REFERENCES

Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, and Charles Sutton. Program synthesis with large language models. *arXiv preprint arXiv:2108.07732*, 2021. URL https://arxiv.org/abs/2108.07732.

Hongtao Chen, Weiyu Xie, Boxin Zhang, Jingqi Tang, Jiahao Wang, Jianwei Dong, Shaoyuan Chen, Ziwei Yuan, Chen Lin, Chengyu Qiu, Yuening Zhu, Qingliang Ou, Jiaqi Liao, Xianglin Chen, Zhiyuan Ai, Yongwei Wu, and Mingxing Zhang. Ktransformers: Unleashing the full potential of CPU/GPU hybrid inference for MoE models. In *Proceedings of the ACM SIGOPS 31st Symposium on Operating Systems Principles*, pp. 1014–1029, New York, NY, USA, 2025. Association for Computing Machinery. doi: 10.1145/3731569.3764843.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. Evaluating large language models trained on code, 2021. URL https://arxiv.org/abs/2107.03374.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. Training verifiers to solve math word problems, 2021. URL https://arxiv.org/abs/2110.14168.

DeepSeek-AI. DeepSeek-V2: A strong, economical, and efficient mixture-of-experts language model, 2024.

Zhixu Du, Shiyu Li, Yuhao Wu, Xiangyu Jiang, Jingwei Sun, Qilin Zheng, Yongkai Wu, Ang Li, Hai Li, and Yiran Chen. SiDA: Sparsity-inspired data-aware serving for efficient and scalable large mixture-of-experts models. In *Proceedings of Machine Learning and Systems*, volume 6, pp. 224–238, Santa Clara, CA, USA, 2024. MLSys. URL https://proceedings.mlsys.org/paper_files/paper/2024/hash/698cfaf72a208aef2e78bcac55b74328-Abstract-Conference.html.

Artyom Eliseev and Denis Mazur. Fast inference of mixture-of-experts language models with offloading, 2023.

Zhiyuan Fang, Zicong Hong, Yuegui Huang, Yufeng Lyu, Wuhui Chen, Yue Yu, Fan Yu, and Zibin Zheng. Accurate expert predictions in MoE inference via cross-layer gate, 2025.

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.

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, pp. 288–304, Miami Beach, FL, USA, 2023. MLSys. URL https://proceedings.mlsys.org/paper_files/paper/2023/hash/5a54f79333768effe7e8927bcccffe40-Abstract-mlsys2023.html.

Georgi Gerganov. llama.cpp: LLM inference in C/C++. https://github.com/ggerganov/llama.cpp, 2023.

Mor Geva, Daniel Khashabi, Elad Segal, Tushar Khot, Dan Roth, and Jonathan Berant. Did aristotle use a laptop? a question answering benchmark with implicit reasoning strategies. *Transactions of the Association for Computational Linguistics*, 9:346–361, 2021. doi: 10.1162/tacl_a_00370.

Jiaao He, Jidong Zhai, Tiago Antunes, Haojie Wang, Fuwen Luo, Shangfeng Shi, and Qin Li. FasterMoE: Modeling and optimizing training of large-scale dynamic pre-trained models. In *Proceedings of the 27th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, pp. 120–134, New York, NY, USA, 2022. Association for Computing Machinery. doi: 10.1145/3503221.3508418.

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, pp. 269–287, Miami Beach, FL, USA, 2023. MLSys. URL https://proceedings.mlsys.org/paper_files/paper/2023/hash/5616d34cf8ff73942cfd5aa922842556-Abstract-mlsys2023.html.

Ranggi Hwang, Jianyu Wei, Shijie Cao, Changho Hwang, Xiaohu Tang, Ting Cao, and Mao Yang. Pre-gated MoE: An algorithm-system co-design for fast and scalable mixture-of-expert inference. In *Proceedings of the 51st Annual International Symposium on Computer Architecture*, 2024. URL https://www.microsoft.com/en-us/research/uploads/prod/2024/05/isca24_pregated_moe_camera_ready.pdf.

Albert Q. Jiang, Alexandre Sablayrolles, Antoine Roux, Arthur Mensch, Blanche Savary, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Emma Bou Hanna, Florian Bressand, Gianna Lengyel, Guillaume Bour, Guillaume Lample, Lélio Renard Lavaud, Lucile Saulnier, Marie-Anne Lachaux, Pierre Stock, Sandeep Subramanian, Sophia Yang, Szymon Antoniak, Teven Le Scao, Théophile Gervet, Thibaut Lavril, Thomas Wang, Timothée Lacroix, and William El Sayed. Mixtral of experts, 2024.

Keisuke Kamahori, Tian Tang, Yile Gu, Kan Zhu, and Baris Kasikci. Fiddler: CPU-GPU orchestration for fast inference of mixture-of-experts models. In *International Conference on Learning Representations*, Singapore, 2025. OpenReview.net. URL https://openreview.net/forum?id=N5fVv6PZGz.

Pingzhi Li, Zhenyu Zhang, Prateek Yadav, Yi-Lin Sung, Yu Cheng, Mohit Bansal, and Tianlong Chen. Merge, then compress: Demystify efficient SMoE with hints from its routing policy. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://arxiv.org/abs/2310.01334.

Jiawei Liu, Chunqiu Steven Xia, Yuyao Wang, and Lingming Zhang. Is your code generated by ChatGPT really correct? rigorous evaluation of large language models for code generation. In *Advances in Neural Information Processing Systems*, 2023. URL https://openreview.net/forum?id=1qvx610Cu7.

NVIDIA. Getting started with CUDA graphs. https://developer.nvidia.com/blog/cuda-graphs/, 2019.

Qwen Team. Qwen3 technical report, 2025.

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

Yun Wang, Lingyun Yang, Senhao Yu, Yixiao Wang, Ruixing Li, Zhixiang Wei, James Yen, and Zhengwei Qi. BuddyMoE: Exploiting expert redundancy to accelerate memory-constrained mixture-of-experts inference, 2025.

Juntong Wu, Jialiang Cheng, Fuyu Lv, Dan Ou, and Li Yuan. SERE: Similarity-based expert re-routing for efficient batch decoding in MoE models. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=98IxaUQtMY.

Leyang Xue, Yao Fu, Zhan Lu, Luo Mai, and Mahesh Marina. MoE-Infinity: Efficient MoE inference on personal machines with sparsity-aware expert cache, 2025.

Enda Yu, Dezun Dong, Zhaoning Zhang, Zhe Bai, Weiling Yang, Haojie Wang, Dongsheng Li, Yongwei Wu, and Xiangke Liao. LayerScope: Predictive cross-layer scheduling for efficient multi-batch MoE inference on legacy servers. In *Proceedings of the 2026 International Conference on Supercomputing*, 2026. doi: 10.1145/3797905.3807834. URL https://arxiv.org/abs/2509.23638.

Shuzhang Zhong, Yanfan Sun, Ling Liang, Runsheng Wang, Ru Huang, and Meng Li. Hybrimoe: Hybrid CPU-GPU scheduling and cache management for efficient MoE inference. In *Proceedings of the 62nd ACM/IEEE Design Automation Conference*, pp. 1–7, San Francisco, CA, USA, 2025. IEEE. doi: 10.1109/DAC63849.2025.11133274.

A SUPPLEMENTARY METHOD DETAILS

DyS-MoE builds on KTransformers model loading, dense operators, quantized weights, and the CPU expert backend. Exact computation uses the originally requested expert with the same weight representation as the reference model. DeepSeek's shared expert follows a separate exact path.

A.1 IMPORTANCE-AWARE MISS HANDLING

Algorithm 1 specifies GPU exact admission, bounded CPU exact selection, and residual replacement. The stages assign each request to one execution path while preserving its original router weight.

Algorithm 1. Importance-Aware Miss Handling

Input: assignments $\mathcal{A}$, available experts $\mathcal{V}$, experts $\mathcal{G} \subseteq \mathcal{V}$ bound to execution slots, slot capacities c, candidate lists L, CPU-priority table, and budget B_{cpu}.
Output: GPU bindings P, CPU exact assignments Q, and unassigned set $\mathcal{U}$.

$P \leftarrow \emptyset$; initialize remaining capacities from c
Compute $I(a)$ using Equation 3
for each $e \in \mathcal{G}$ **do**
  $P \leftarrow P \cup \text{Admit}(\{a \in \mathcal{A} : e_a = e\}, e; I)$
$\mathcal{M} \leftarrow \mathcal{A} \setminus \text{Assigned}(P)$
Compute $S(a)$ from the remaining capacities using Equation 3
$Q \leftarrow \text{TopEligible}(\mathcal{M}, S, B_{\text{cpu}})$
$\mathcal{U} \leftarrow \mathcal{M} \setminus Q$
for candidate rank $k = 1, \ldots, K$ **while** $\mathcal{U} \neq \emptyset$ **do**
  for each $j \in \mathcal{G}$ **do**
    $V \leftarrow \{a \in \mathcal{U} : L[e_a, k] = j\}$
    $P \leftarrow P \cup \text{Admit}(V, j; I)$
  $\mathcal{U} \leftarrow \mathcal{U} \setminus \text{Assigned}(P)$
return $P, Q, \mathcal{U}$

$\text{Admit}(V, e; I)$ binds the highest-I requests that fit expert e's remaining capacity and decrements that capacity, preserving router weights. TopEligible selects at most B_{cpu} eligible requests by S; $L[e, k]$ is source e's kth candidate, excluding e itself, and K is the maximum candidate-list length. Missing candidates make no proposal. Complete allocation has $\mathcal{U} = \emptyset$.

CPU eligibility and ordering. Only active assignments remaining after GPU exact admission are eligible. TopEligible selects up to B_{cpu} such assignments with $S(a) \geq \tau_{\text{cpu}}$; the default threshold is zero. Requests are ordered by decreasing S, then decreasing I, and finally by token index and router rank. GPU admission uses decreasing I with the same index-based tie breaking. No router rank is unconditionally reserved for CPU computation in the evaluated configuration.

Slot binding and capacity. The available set $\mathcal{V}$ contains resident and ready-prefetched experts; the execution set $\mathcal{G} \subseteq \mathcal{V}$ contains those bound to slots. CPU priority considers experts in $\mathcal{V}$ with execution capacity remaining after initial routing admission; GPU admission and replacement operate over $\mathcal{G}$. When cache slots outnumber execution slots, binding fills larger slots first. Experts are ranked by their summed assignment importance, weighted by the fraction of demand that fits. Unused slots retain available experts as replacement targets. Binding changes indices rather than moving weights. The number of slots in each tier is proportional to its capacity, rounded to the total slot count; priority slots receive larger tiers first.

The capacity template is selected before graph capture and remains fixed during replay. Appendix B.3 gives the capacity and candidate-coverage conditions under which Algorithm 1 assigns every request.

A.2 SLOTTED GROUPED EXECUTION

Persistent buffers store expert-slot bindings, token indices, router weights, and valid assignment counts. The grouped operators gather admitted rows, fuse gate/up projections with SwiGLU, and

apply the down projection followed by weighted output scatter. GPU exact and replacement assignments share this path; their bindings determine which expert weights are used. Capacity tiers organize long-tailed demand within fixed buffers. Operators process admitted rows, so reduced reservation and reduced execution time are distinct benefits. CPU input/output buffers and synchronization dependencies also remain fixed across replays.

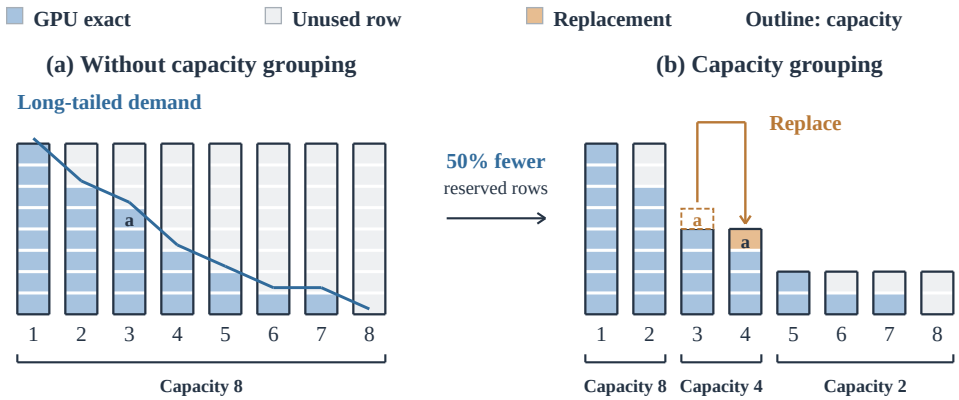


Figure 5: Capacity grouping halves reserved capacity (64 to 32 rows), reducing unused padding by 84.2% (38 to 6 rows) for the same workload. Overflow assignment a uses expert 4's spare capacity through replacement.

A.3 Execution-Window Dependencies

The execution windows in Figure 3 preserve three dependencies. First, the GPU binding stage in W_i prepares descriptors for MoE_{i+1}. An expert prefetched for layer $i+2$ must therefore be published before binding in W_{i+1}. Second, selected weights remain unchanged until the corresponding computation completes. Third, CPU exact outputs are merged into the current MoE output before next-layer attention. Host preparation separates consecutive replays; boundary windows handle the start and end of the network. These dependencies preserve the execution structure while allowing expert bindings to change and CPU computation and prefetch to overlap GPU work.

A.4 Execution Consistency

Graph replay and direct execution of the same prepared windows produce matching logits, greedy tokens, and active assignment descriptors across eight teacher-forced steps on both models, including partially active batches of size 8. These checks cover disabled prefetch publication and identical caches with a fixed nonempty prefetch-ready state. They test replay consistency under matched inputs and state; throughput measurements use live prefetch.

A.5 Assignment Admission

Eight teacher-forced steps from one full task batch of size 8 provide a local view of admission. Across 208 DeepSeek layer-steps, 6,323 of 9,984 assignments request GPU-available experts, and 6,220 execute as GPU exact. Across 384 Qwen layer-steps, the corresponding counts are 21,657 of 24,576 and 21,465. GPU-available assignments that do not execute as GPU exact account for 1.03% and 0.78% of all assignments, respectively. This quantity includes both binding and capacity admission effects; it is not a full-task estimate of capacity overflow alone.

B Offline Calibration

Offline preparation combines model calibration with hardware-aware budget profiling. For GSM8K evaluation, we use 24 prompts from its training split. For HumanEval evaluation, we calibrate all model-dependent artifacts on 24 prompts selected from MBPP+ (Austin et al., 2021; Liu et al.,

2023). Calibration runs follow the exact-expert path with at most 128 generated tokens per prompt, supplying routing traces and expert-output statistics for the resident configuration, replacement and CPU priority tables, and prefetch predictor.

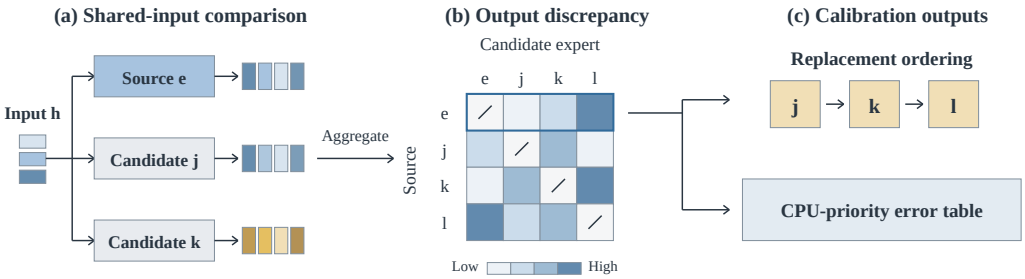


Figure 6: Expert-output calibration. Candidate experts process shared inputs; aggregated output discrepancies determine replacement ordering and inform CPU exact priority.

B.1 Replacement Error and CPU Priority

Equation 5 compares expert outputs on exact-model states where the source expert was routed. RMS is computed over output dimensions, with $\epsilon = 10^{-6}$. The statistic is source-conditioned: $D_\ell(e, j)$ need not equal $D_\ell(j, e)$. Candidate lists rank replacement experts by this discrepancy. CPU selection uses a separately calibrated table D_ℓ^{cpu}, computed with the same normalized output-error statistic. At CPU selection, $\mathcal{V}$ contains currently available resident and ready-prefetched experts; pending transfers are excluded. The pairwise errors are calibrated offline, while the minimum is restricted to candidates with spare execution capacity after GPU exact admission of the initial routed requests:

$$E_\ell^{\text{cpu}}(e; \mathcal{V}) = \alpha_{\ell,e} \min_{\substack{j \in \mathcal{V} \setminus \{e\} \\ c_j^{\text{route}} > 0}} D_\ell^{\text{cpu}}(e, j). \tag{6}$$

Here c_j^{route} is the remaining execution capacity for expert j after initial routing admission, and is zero for experts without a bound execution slot. CPU selection precedes residual replacement, so its scores use this capacity snapshot and are not recomputed as replacements consume capacity. Excluding the source also handles overflow of a resident expert's slot. The source multiplier $\alpha_{\ell,e}$ is its root-mean-square output magnitude over calibration inputs, divided by the median magnitude of sufficiently observed sources in the same layer. Sources with fewer than eight observations retain multiplier one. This scaling changes CPU priority, not replacement ordering or output router weights.

Calibrated candidate pairs precede unmeasured pairs; ties in calibrated error are resolved by expert ID. When no calibrated error or similarity is available, candidates are ordered by expert-ID distance to the source, with stable slot order breaking ties. Missing CPU pair estimates fall back to the replacement-error table; a resident source without a calibrated alternative receives unit error, while an unavailable source with no finite estimate receives highest CPU priority. These rules retain executable candidates when calibration coverage is incomplete.

B.2 Prefetch Prediction and Resident Selection

The predictor projects current-layer hidden states with the known target-layer gate weights, then applies offline score calibration and routing-transition statistics from the current routing information. It ranks experts per token, aggregates demand across the batch, and selects prefetch candidates. It uses no future-layer activations and only guides prefetching; the model's actual router is unchanged. Predictor fitting and validation reuse the offline calibration corpus, augmented with sampled hidden states, current routing information, and future expert-selection labels from the same exact-path traces. DeepSeek additionally uses a rank-32 correction fused into its prediction gate before inference, while Qwen uses no low-rank correction. Resident selection uses a calibrated expert ranking

Table 3: Prefetch-predictor size and execution cost at batch size 8 and 40% cache, with CPU budget 1 and prefetch degree 2. Timings cover the complete prediction graph, including gate projection, calibration, and candidate selection.

	Qwen3-30B-A3B	DeepSeek-V2-Lite
Lookahead (layers)	2	1
Input shape	8×2048	8×2048
Expert-score shape	8×128	8×64
Predicted experts per token	8	6
Prediction heads	46	25
Calibration tensor elements	759,782	1,793,725
Calibration artifact size (MiB)	2.90	6.85
GPU execution per prediction (ms)	0.0380	0.0302
Predictor GPU time per step (ms)	0.683	0.726
Decode-step time (ms)	18.154	16.371
Predictor GPU time / step time	3.76%	4.43%
Overlap with main execution windows	99.60%	77.98%

for Qwen and a mixture-energy profile for DeepSeek. Resident experts remain fixed during evaluation; prefetch updates transient slots.

Calibration tensors include expert-transition statistics, so their element counts are not trainable-parameter counts. Artifact sizes exclude runtime storage; DeepSeek's fused prediction gates occupy an additional 6.25 MiB. The predictor runs on a separate CUDA stream. Nsight profiles collected over 64 steps after 128 warmup steps show that its GPU execution accounts for 3.76–4.43% of the same run's decode-step time, with most of that execution overlapping the main windows (Table 3). These measurements indicate a small, largely overlapped prediction workload. Per-step costs reflect actual invocations, since pending prefetch requests can suppress new predictions.

B.3 Capacity Selection

For a batch of b tokens routed to k experts each, the capacity template is chosen to satisfy $\sum_{s \in \mathcal{S}} c_s \geq bk$, where $\mathcal{S}$ contains slots bound to available expert weights. This condition reserves enough GPU capacity without relying on CPU budget utilization. Replacement lists cover all available experts in preference order. A residual request whose original slot is full can therefore use another slot with spare capacity. If a request remained after exhausting its candidates, all usable slots would be full, contradicting the total capacity bound. Thus $\mathcal{U} = \emptyset$; local capacity overflow changes the selected expert rather than dropping an assignment. Grouping redistributes the reserved rows across slots while preserving this total-capacity condition.

B.4 Hardware Profiling and Offline Cost

The same exact-path calibration runs supply routing traces and sampled layer inputs. Evaluating expert outputs on these shared inputs provides source–candidate discrepancies for the error tables, while routing traces supply the labels for prefetch-predictor fitting and validation. Resident selection aggregates the observed expert demand. These stages reuse the calibration data without updating the base model weights.

Hardware profiling measures operation times and inspects the decode-step timeline on the target deployment to identify overlap opportunities and throughput costs as CPU and prefetch budgets vary. The main experiments use a common CPU budget of one and prefetch degree of two for comparison; these settings are not selected to maximize accuracy or throughput on each task. The budget sweeps characterize alternative operating points without requiring exhaustive downstream accuracy tuning during calibration.

The complete offline preparation takes approximately six hours on our evaluation machine, including trace collection, configuration and error-table construction, predictor fitting and validation, and hardware profiling. The resulting artifacts are reused across inference runs; changes to the hardware or execution configuration require rechecking the execution budgets.

C Evaluation Protocols

C.1 System Configurations and Throughput

DyS-MoE uses the batch, cache, and budget settings specified in Section 5.1. The cache sweep reserves four transient weight slots in two banks in addition to the resident set. KTransformers uses CPU experts with graph replay. We extend the released HybriMoE implementation to support batched decoding while retaining its HSS scheduling algorithm. The extension explicitly invokes HSS for multi-row decode inputs, consolidates routing-index construction, and repairs shared-buffer reuse dependencies. Single-row decoding retains the native scheduling entry point. We recalibrate its CPU, GPU, and transfer-time models on our evaluation hardware. Its batch sweep uses 40% residency; the cache sweep varies residency over 20%, 40%, and 60%. Reported throughput uses this adapted implementation. The llama.cpp batch curve uses its default sweep configuration, while the cache comparison uses the corresponding residency settings. These comparisons measure configured systems on shared hardware, not identical memory budgets.

Software versions. The KTransformers baseline uses the archived implementation reporting version 0.4.1, with expert-tail correctness fixes and model compatibility adapters. HybriMoE is based on revision `508a85b`, with the fixes, hardware calibration, and batched scheduling adapter described above. The llama.cpp server binary reports revision `d5003b6`, built with GNU 13.3.0 for Linux x86-64. These identifiers describe the baseline implementations rather than the DyS-MoE development revision.

Throughput excludes warmup and divides generated tokens by measured decode time; llama.cpp uses server-reported decode windows. Controlled component benchmarks repeat the prompt "Explain how a computer cache reduces repeated memory accesses." across batch lanes to control input variation and facilitate repeatable execution-time comparisons. Reported throughput is the arithmetic mean of three runs for each configuration. Main throughput comparisons and component ablations are measured in separate benchmark runs; ablation speedups use the Full DyS-MoE reference measured within the same ablation suite.

C.2 Task Quality and Component Ablations

The original-model reference preserves routed experts and router weights with the same quantization as DyS-MoE. HumanEval uses chat-formatted greedy generation, a 512-token limit, and a 10-second test timeout. The reference uses batch size 1; DyS-MoE uses length-bucketed batches of size 8. GSM8K uses four-shot prompts, a 1,024-token limit, and first-marked-answer scoring. Inactive lanes in incomplete batches are excluded from scoring.

The CPU ablation disables routed-expert CPU exact computation. The prefetch ablation disables prediction, weight transfer, and transient-expert publication while retaining the allocated transient storage and resident identities. Removing capacity grouping uses a uniform capacity template in place of $[8, 4, 2]$. Other model, resident-profile, and risk settings are held fixed within each comparison. Table 8 separately varies CPU and prefetch budgets with the other budget fixed.

C.3 Execution-Time Profiling

Figure 1(a) combines independently measured throughput with Nsight execution profiles. Profiling uses 128 warmup steps and 64 captured steps. KT and DyS traces include graph-internal nodes; HybriMoE uses a non-graph trace. The HybriMoE profile uses 16 resident experts, whereas the DyS profile uses 40% residency, CPU budget 1, and prefetch degree 2. CPU-only, GPU-only, CPU–GPU overlap, and runtime intervals form the time-share breakdown, counting overlap once. Runtime denotes execution organization and data movement outside model computation. Instrumented timings are used for this breakdown, not for throughput comparisons.

D Supplementary Experimental Results

D.1 Throughput across Batch Sizes

Table 4: Decode throughput (tokens/s) corresponding to Figure 4.

Batch size	KTransformers	HybriMoE	llama.cpp	DyS-MoE
Qwen3-30B-A3B				
1	30.26	4.94	31.73	78.15
2	48.75	9.69	63.73	133.52
4	75.40	19.30	98.18	238.72
6	90.10	24.26	116.25	350.56
8	86.68	27.11	123.67	431.99
10	124.57	30.97	131.52	489.70
12	113.25	31.32	134.63	571.09
14	125.42	17.66	138.66	612.61
16	145.49	47.87	140.02	617.46
DeepSeek-V2-Lite				
1	43.33	9.55	33.89	73.29
2	64.35	15.26	52.57	143.20
4	95.88	28.90	62.15	273.81
6	114.21	39.81	66.52	449.60
8	79.88	42.95	67.42	551.06
10	124.47	37.61	70.41	719.85
12	137.92	44.01	66.38	819.21
14	139.73	40.18	67.83	866.61
16	148.20	52.91	68.76	1066.98

D.2 Task-Quality Counts

The quality evaluations contain 1,319 GSM8K problems and 164 HumanEval problems. The original-model HumanEval references pass 147 and 57 problems on Qwen and DeepSeek, respectively.

Table 5: Correct / evaluated problems for the component ablations in Table 2.

Model	Configuration	GSM8K	HumanEval
Qwen	Full DyS-MoE	1226 / 1319	143 / 164
	w/o capacity grouping	1222 / 1319	143 / 164
	w/o CPU compute	1212 / 1319	139 / 164
	w/o prefetch	1227 / 1319	145 / 164
DeepSeek	Full DyS-MoE	733 / 1319	54 / 164
	w/o capacity grouping	721 / 1319	52 / 164
	w/o CPU compute	669 / 1319	36 / 164
	w/o prefetch	724 / 1319	56 / 164

Table 6: Correct / evaluated problems for the cache-ratio comparison in Table 1.

Model	Task	20%	40%	60%
Qwen	GSM8K	1181/1319	1226/1319	1222/1319
	HumanEval	143/164	143/164	146/164
DeepSeek	GSM8K	563/1319	733/1319	763/1319
	HumanEval	33/164	54/164	61/164

D.3 Miss-Handling Strategies

Importance-Aware Miss Handling answers 1,226/1,319 GSM8K problems and 143/164 HumanEval problems correctly. The randomized policy answers 1,188/1,319 and 125/164 correctly, respectively. We port the expert-related offline procedures of BuddyMoE and SERE (Wang et al., 2025; Wu

et al., 2026) and use their respective strategies to determine expert-replacement priorities within the DyS-MoE execution framework. All three strategies use the same set of 24 MBPP+ calibration prompts. On the complete 164-problem HumanEval set, with batch size 8, 40% cache, CPU budget 1, and prefetch degree 2, they solve 138 and 142 problems, respectively, compared with 143 for DyS-MoE. This comparison evaluates the transferred offline preparation and replacement-priority policies within our runtime, rather than the complete external inference systems.

Table 7: Miss-handling policies on Qwen at batch size 8 and 40% cache, with CPU budget 1 and prefetch degree 2. Parentheses give changes relative to Importance-Aware Miss Handling, in percentage points. The randomized policy randomizes both CPU selection and GPU replacement. BuddyMoE and SERE rows use their offline procedures and replacement-priority policies within the DyS-MoE runtime; dashes indicate unevaluated tasks.

Policy	GSM8K (%)	HumanEval (%)
Importance-Aware Miss Handling	92.95	87.20
Randomized miss handling	90.07 (−2.88)	76.22 (−10.98)
BuddyMoE-based replacement	—	84.15 (−3.05)
SERE-based replacement	—	86.59 (−0.61)
Original model	92.34 (−0.61)	89.63 (+2.44)

D.4 CPU and Prefetch Budget Selection

Table 8: CPU and prefetch budget trade-offs on Qwen at batch size 8 and 40% cache. Each sweep fixes the other budget. Throughput is in tokens/s and HumanEval accuracy is in %.

CPU budget (prefetch = 2)			**Prefetch budget** (CPU = 1)		
Budget	Tokens/s	HumanEval	Budget	Tokens/s	HumanEval
0	389.95	84.76	0	374.52	88.41
1	364.81	87.20	1	369.07	89.63
2	334.87	88.41	2	364.81	87.20
3	267.73	88.41	3	341.92	87.80
			4	260.18	90.24

Table 8 examines how much exact work the pipeline can accommodate. Raising the CPU budget from zero to two improves HumanEval from 84.76% to 88.41%, while reducing throughput from 389.95 to 334.87 tokens/s. A third assignment provides no further accuracy gain and lowers throughput to 267.73 tokens/s, indicating diminishing returns from additional CPU work.

Prefetch likewise exposes a quality–efficiency trade-off. Degree four reaches 90.24% accuracy at 260.18 tokens/s, whereas degree one achieves 89.63% at 369.07 tokens/s. Accuracy is not monotonic in prefetch degree, consistent with changes in both expert availability and replacement choices. Prefetch is therefore an optional cache-update mechanism rather than a prerequisite for window replay. Budget selection includes degree zero and uses hardware profiling to limit transfer costs; task-level evaluation establishes the resulting quality trade-off.

D.5 GPU Memory Footprint

Table 9 compares process GPU memory for Qwen3-30B-A3B at batch size 8 with 1,024 input tokens, measured during decode steps 129–256. Matching resident expert counts does not imply matching memory footprints: DyS-MoE preserves the original GGUF expert weights, with Q4_K gate/up projections and down projections in Q4_K for 24 layers and Q6_K for the other 24 layers. HybriMoE requantizes GPU expert weights into its 4-bit Marlin representation.

The larger mixed-format weight payload accounts for a substantial portion of DyS-MoE's additional memory, and the gap grows with residency. DyS-MoE also reserves graph execution buffers and

Table 9: Qwen3-30B-A3B GPU memory at batch size 8 (GiB). Weight payload denotes resident expert weight data rather than the complete process allocation.

	Process memory			Resident weight payload	
Cache ratio	DyS-MoE	HybriMoE	Difference	DyS-MoE	HybriMoE
20%	7.699	6.643	+1.056	3.321	2.742
40%	11.051	9.279	+1.772	6.514	5.379
60%	14.486	12.021	+2.465	9.834	8.121

transient prefetch storage. Our choice to preserve GGUF weights controls the sources of numerical change: it avoids an additional weight-quantization stage and lets CPU and GPU expert paths use the same quantized weights. HybriMoE's Marlin path instead emphasizes GPU execution and storage efficiency. The DyS-MoE execution mechanism does not require GGUF and could accommodate requantized GPU experts, although their quality–memory trade-off is not evaluated here. Thus these memory differences reflect both weight representation and execution storage, rather than an intrinsic requirement of window replay. DyS-MoE targets high decode throughput under a GPU memory constraint, rather than minimum memory use alone. Cache capacity controls the available expert set and the resulting quality–throughput trade-off: smaller caches reduce storage but can require more replacement, while larger caches offer more exact-execution opportunities at a greater memory cost.

KTransformers places expert computation on the CPU and has no comparable GPU expert-cache ratio. Under the Qwen measurement configuration above, its sampled peak and steady-state mean GPU memory are 2.729 and 2.443 GiB, respectively. This lower GPU memory footprint trades GPU-resident expert execution for CPU expert computation; DyS-MoE uses more GPU memory to obtain the higher decode throughput reported in Figure 4.

D.6 Calibration Transfer

Cross-dataset calibration. Using 24 MBPP+ calibration prompts to calibrate all model-dependent artifacts, Qwen achieves 87.20% on HumanEval (143/164), compared with 89.63% (147/164) for the original model, a difference of four solved problems. DeepSeek achieves 32.93% (54/164), compared with 34.76% (57/164), a difference of three problems. Both use batch size 8, 40% cache, CPU budget 1, and prefetch degree 2. These results show that the resident configuration, replacement statistics, and prefetch predictor can be calibrated on a separate code dataset while retaining accuracy close to the exact-expert reference on HumanEval.

Transfer to StrategyQA. We evaluate Qwen on all 229 questions in the StrategyQA development set (Geva et al., 2021), using batch size 8, 40% cache, CPU budget 1, and prefetch degree 2. We use zero-shot chat prompts requesting a brief explanation ending with `Answer: yes/no`, with thinking disabled and greedy decoding capped at 128 tokens. Scoring requires exactly one standalone `Answer: yes` or `Answer: no` line (case-insensitive, with an optional trailing period or exclamation mark); missing or multiple matches are scored as incorrect. Calibration uses source-task prompts without StrategyQA examples (Table 10). GSM8K-only calibration achieves 63.76% accuracy; adding 24 MBPP+ prompts raises it to 65.50%, eight correct answers below the exact reference. This demonstrates transfer to a reasoning benchmark absent from calibration. The mixed-source configuration changes both calibration size and composition, so the gain is not attributed to source diversity alone.

Table 10: Qwen calibration transfer to StrategyQA. Accuracy differences are in percentage points relative to the KTransformers exact-expert reference.

Calibration / reference	Correct	Accuracy (%)	Difference
KTransformers exact	158/229	69.00	—
GSM8K (24 prompts)	146/229	63.76	−5.24
GSM8K + MBPP+ (24 each)	150/229	65.50	−3.49

Cross-task resident selection. With a resident profile calibrated on GSM8K routing traces, DeepSeek-V2-Lite achieves 30.49% on HumanEval (50/164), compared with 34.76% (57/164) for the original model with exact expert computation, a gap of 4.27 percentage points. DyS-MoE uses batch size 8, 40% resident cache, CPU budget 1, and prefetch degree 2. This result demonstrates cross-task applicability of the resident profile, from mathematical reasoning calibration to code generation evaluation.