MoEAtlas: Orchestrating Communication in MoE Inference via Pre-Routing, Expert Placement and Migration

Anonymous authors
Paper under double-blind review

Abstract

DeepSeek reports substantial communication overhead in mixture-of-experts (MoE) inference with expert parallelism across nodes. Each MoE layer sends tokens to GPUs holding their selected experts. Placing experts selected together on fewer nodes reduces transfers; choosing where each request runs makes more transfers local. We present MoEAtlas, which coordinates Pre-Router, hypergraph expert placement, and online migration to reduce communication in distributed MoE inference. The offline-trained Pre-Router predicts expert selections of prompt tokens across MoE layers and uses these predictions with the current expert layout to select a home GPU for each request. We represent each token's co-activated experts as a hyperedge and use hypergraph partitioning algorithms to produce new expert placements to reduce inter-node communication under load and GPU memory budgets. As workloads change, our migration strategy optimizes the mapping of the target expert placement to devices to reduce migration volume and time while inference continues. New hypergraph expert layouts with Pre-Router reduce complete prefill time by 19.33 %, 17.00 %, and 24.69 % on OLMoE, Qwen1.5-MoE, and DeepSeek-V2-Lite, respectively, compared with the original layouts. On OLMoE, min-volume mapping reduces inter-node weight-transfer volume by 11.95 % relative to fixed mapping; online migration reduces the total time to serve requests and complete a layout update by 6.83 % relative to blocking migration.

1 Introduction

Mixture-of-experts (MoE) models increase parameter capacity by activating a subset of experts per token and distributing expert weights across GPUs (Shazeer et al., 2017; Fedus et al., 2022; Muennighoff et al., 2025). Expert parallelism requires token dispatch and output combination, making both load balance and communication cost central to inference (Rajbhandari et al., 2022; Hwang et al., 2023; Li et al., 2023). DeepSeek's inference report identifies substantial cross-node communication overhead and describes overlapping communication with computation (DeepSeek-AI, 2025a).

A request's inter-node communication depends on where its experts reside and where the request executes. Grouping frequently co-activated experts can reduce the number of destination nodes; choosing a suitable request home makes some of these visits local. The decisions are coupled: moving experts changes which home is favorable, while evolving request demand can change which expert groups are useful.

With node-coalesced communication, a token is sent once to each remote node hosting its selected experts. Two selected experts on the same remote node share one transfer; placing them on different remote nodes requires two. The full co-activation set and its placement determine the destination nodes. These combinations can change as the workload evolves, making a previously effective grouping less suitable for reducing communication. Monitoring co-activation patterns helps identify when the expert layout should be updated.

We present **MoEAtlas**, which coordinates pre-routing, expert placement, and migration (Figure 1). An offline-trained Pre-Router predicts prompt-token expert sets and uses the active layout and a node-aware cost model to select a home GPU. Persistent co-activation drift triggers hypergraph-based

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.

expert placement under load and memory limits. After an execution-time check, online migration gradually moves from the old layout to the new one while inference continues.

Our contributions are:

- **Pre-routing for sequence-level scheduling** (Section 4.1). An offline-trained parallel predictor estimates per-layer expert sets; Pre-Router uses the active layout, node-aware communication cost, and admission constraints to select request homes.
- **Communication-aware hypergraph placement** (Section 4.2). We represent complete co-activation sets as weighted hyperedges, minimize node visits under per-layer load limits, and map partitions to memory-feasible GPU layouts.
- **Online expert migration** (Section 4.3). Given an accepted layout, overlap-aware node and GPU matching retains resident weights; parallel transfer rounds and safe handoffs install it while inference continues.

On fixed OLMoE layouts, Pre-Router reduces inter-node token sends by 8.29 % versus count-balanced homes. Across three models, MoEAtlas reduces complete prefill time by 17.00–24.69 % versus load-only placement with count-balanced homes. On OLMoE, unrestricted online migration reduces complete time for 512 prefills and migration by 6.83 % versus blocking migration.

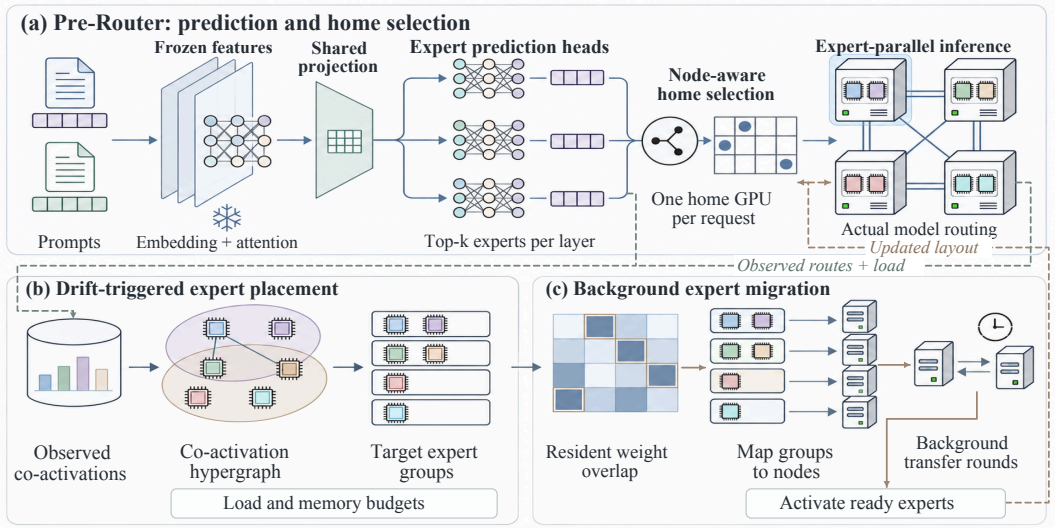

Figure 1: MoEAtlas overview. **(a)** Predicted expert sets guide request homes; the model's router remains unchanged. **(b)** Observed co-activation drift triggers constrained expert placement. **(c)** After an MoE execution-time check, parallel weight transfers update the layout while serving continues. Dashed paths show observed demand and expert-map updates.

2 Related Work

Request scheduling. ELDR groups requests by expert signatures observed during prefill on decode workers, reducing distinct experts per batch (Choi et al., 2026); Bambhaniya et al. (2026) route activation-aware decode micro-batches to expert groups. Sem-MoE coordinates offline expert placement and online request/token scheduling through token-ID–expert activation statistics (Li et al., 2026), which cannot distinguish context-dependent routes for the same token ID. MoEAtlas predicts prompt-token top-k sets before prefill, chooses a home GPU under the current layout, and updates placement after observed co-activation drift.

Expert placement. EPLB balances load through replication and packing, with predefined expert groups for locality (DeepSeek-AI, 2025b). Occult and CAP use pairwise co-activation (Luo et al., 2025; Shi et al., 2026); GRACE-MoE combines spectral grouping, replication, and locality-aware routing (Han et al., 2025). For $k > 2$, pair counts can hide different complete activation sets and

node-visit costs (Figure 2a). Sem-MoE counts remote expert activations individually (Li et al., 2026); MoETuner optimizes inter-GPU token flows between expert pairs in adjacent layers (Go & Mahajan, 2025). Director predicts queued-request routes to update expert placement while overlapping migration with computation (Liu et al., 2026); its workload fixes each request's source GPU, whereas MoEAtlas chooses the request home for a given layout. MoEAtlas represents complete top-k sets as hyperedges, minimizes node visits under load constraints, and checks GPU memory during device mapping, capturing shared transfers to multiple experts on one remote node.

3 Communication and Computation Costs

Consider L MoE layers on M nodes and P GPUs. Layout x places expert e in layer l on GPU $g_l(e)$ and node $x_l(e) = \mu(g_l(e))$; routed experts have one serving location. The model router selects the top-k set $A_{t,l}$ for token t. Request s has T_s prompt tokens and stores its KV cache on home GPU c_s, on node $h_s = \mu(c_s)$.

Communication volume. Node-coalesced dispatch sends a token once to each remote node hosting a selected expert; combine returns one output per visited node. Let $D_{t,l}(x)$ be the visited nodes, $F_s(x)$ their total visits across request s, and $a_s(m; x)$ visits to node m:

$$D_{t,l}(x) = \{x_l(e) : e \in A_{t,l}\}, \qquad F_s(x) = \sum_{t \in s,l} |D_{t,l}(x)|, \qquad a_s(m; x) = \sum_{t \in s,l} \mathbf{1}[m \in D_{t,l}(x)]. \tag{1}$$

Subtracting local visits gives the inter-node dispatch payload count:

$$Q_s(h_s; x) = F_s(x) - a_s(h_s; x) = \underbrace{F_s(x) - T_s L}_{\text{expert grouping}} + \underbrace{T_s L - a_s(h_s; x)}_{\text{extra transfers from the request home}}. \tag{2}$$

The first term measures extra destination nodes, reduced by grouping co-activated experts; the second measures visits missed by the request home. Endpoint byte counts estimate dispatch and combine time, whose layerwise sum is $\widehat{T}_{\text{comm}}$. Let $\hat{t}^{\text{exp}}_{l,g}$ be GPU g's expert computation time from its token counts. The batch score adds the peak GPU time in each layer:

$$\widehat{T}_{\text{MoE}}(x, \mathbf{c}; A) = \widehat{T}_{\text{comm}}(x, \mathbf{c}; A) + \sum_{l=1}^{L} \max_g \hat{t}^{\text{exp}}_{l,g}(x; A). \tag{3}$$

Pre-Router compares homes using $\widehat{T}_{\text{comm}}$; layout selection uses $\widehat{T}_{\text{MoE}}$. Appendix A.2 details the endpoint estimator and stage aggregation.

4 MoEAtlas

Pre-Router predicts expert selections and chooses request homes under the active expert layout. Persistent co-activation drift triggers hypergraph-based expert placement. Online migration gradually moves from the old layout to the new one while inference continues. We describe these components below; Appendix C.4 gives the complete online loop.

4.1 Pre-Router

Pre-Router predicts expert selections across all MoE layers and uses them with the active expert layout to select each request's home GPU before admission. Cache-aware routing fixes the execution GPU for prompts with reusable KV cache, whose migration would be costly; home selection therefore targets cache-miss prefills. All prompt tokens are available to Pre-Router. Its predictor has a frozen frontend that reuses the target model's pretrained embedding, input normalization, and first self-attention sublayer. Given prompt embeddings $H^{(0)} = (h_t^{(0)})_{t \in s}$,

$$u_t = h_t^{(0)} + \left[\text{Attn}_0(\text{Norm}_0(H^{(0)}))\right]_t, \qquad z_t = \text{LN}(W_p u_t + b). \tag{4}$$

where $[\cdot]_t$ selects token t's output. The shared projection maps $u_t \in \mathbb{R}^H$ to $z_t \in \mathbb{R}^d$. It is computed once per token, shared by all parallel heads, and trained together with them offline.

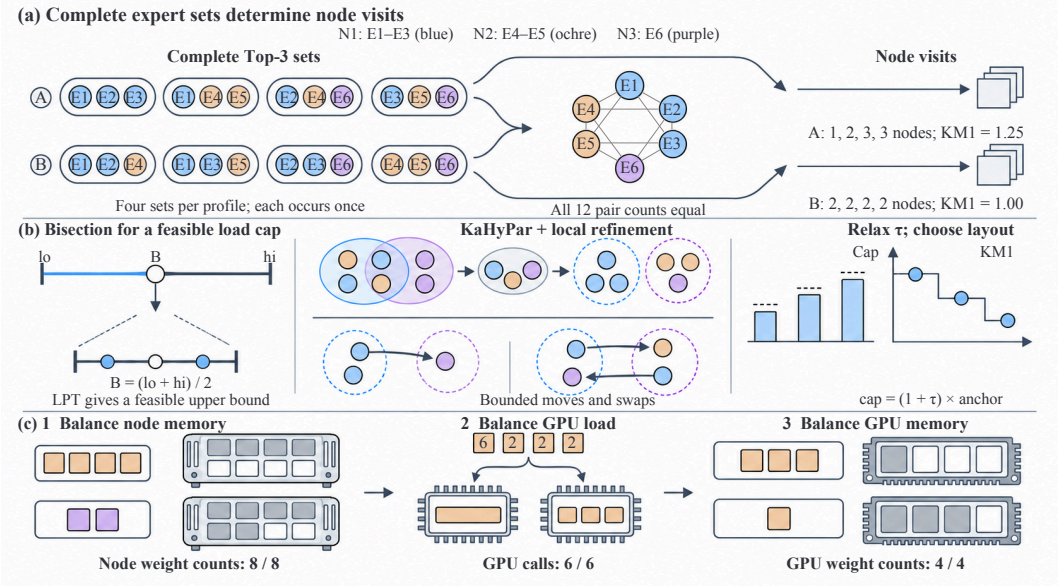

Figure 2: Hypergraph expert placement. **(a)** Complete Top-3 sets reveal different node visits and KM1 with equal pairwise counts. **(b)** Bisection finds a feasible load anchor; KaHyPar and local refinement reduce KM1 as τ relaxes the load cap. **(c)** Mapping balances node weight storage, GPU load, and GPU weight storage. Gray slots hold earlier-layer weights.

Parallel prediction heads. Each parallel head f_l has its own two linear layers, $d \to h$ and $h \to E$, with GELU between them. Here h is the head width and E is the number of experts:

$$q_{t,l} = \mathrm{softmax}(f_l(z_t)), \qquad \mathcal{L} = \sum_l \alpha_l \, \mathbb{E}_t \mathrm{KL}(p_{t,l} \| q_{t,l}), \tag{5}$$

Here $p_{t,l}$ is the true router distribution used for offline supervision, with $\alpha_l = 1$ for all layers.

The top-k entries of $q_{t,l}$ give $\widehat{A}_{t,l}$ for Pre-Router's home selection. We report prediction quality and full prediction time, including the frontend, in Section 5.2. Architecture and training details are in Appendix A (Figure 5).

Selecting request homes. Using predicted routes and the active layout, Pre-Router assigns requests in arrival order to the feasible home minimizing estimated batch communication $\widehat{T}_{\mathrm{comm}}(x, \mathbf{c}; \widehat{A})$ (Equation (13)), accounting for earlier assignments. The home persists through prefill and decoding; Appendix A.2 gives the selection details.

4.2 Hypergraph Expert Placement

MoEAtlas retains each complete top-k expert set as an occurrence-weighted hyperedge, preserving node-visit information. Appendix B.1 compares pairwise and per-expert statistics.

A co-activation hypergraph. We construct one hypergraph per MoE layer, with one vertex (l, e) for expert e in layer l. Each hyperedge a contains a complete set of jointly selected experts; its weight $w_{l,a}$ counts how often that set appears in the profiling window. The resulting number of calls to expert e is $C_{l,e} = \sum_{a: e \in a} w_{l,a}$.

Let $b_l(e) \in \{1, \ldots, M\}$ denote the logical node group containing expert e in layer l. We optimize group membership under a load limit per group:

$$\begin{aligned} \min_{b} \quad & \sum_{l,a} w_{l,a} \big(|\{b_l(e) : e \in a\}| - 1 \big), \\ \text{subject to} \quad & \sum_{e: b_l(e) = j} C_{l,e} \le B_l(\tau), \qquad \forall l, j. \end{aligned} \tag{6}$$

For each activation set, the objective counts its visited nodes minus one, weighted by its occurrence count. This is the connectivity-minus-one (KM1) cost, corresponding to the grouping term in Equation (2). The constraint limits expert calls per node and layer. We set $B_l(\tau) = (1+\tau)\widehat{B}_l$, where $\widehat{B}_l$ is a feasible peak-load anchor found separately for each layer. All layers share the same $\tau \geq 0$, which controls the allowed fractional load increase relative to their respective anchors. At $\tau = 0$, each limit equals its anchor; larger τ gives more freedom to reduce communication.

Hypergraph partitioning under load constraints. For a given node-load limit, KaHyPar generates candidate partitions using weighted KM1; feasible candidates receive bounded local refinement. Appendix B.2 gives the move, swap, and solver details.

Finding a load anchor and relaxing the limit. Figure 2b shows the search. For each layer l, the larger of the average node load and the busiest expert's load gives a lower bound on peak node load:

$$B_l^{\mathrm{LB}} = \max\left\{\frac{1}{M}\sum_e C_{l,e},\ \max_e C_{l,e}\right\} \leq \widehat{B}_l. \tag{7}$$

Longest-processing-time-first (LPT) placement gives a feasible initial upper bound: it assigns experts in decreasing load order to the least-loaded node, without limiting expert count (Graham, 1969). Its peak load is at least B_l^{LB}. Between these bounds, we bisect the proposed load cap and run KaHyPar with several seeds. A feasible partition lowers the upper bound; if none is found, the search raises the lower probe bound. This heuristic search yields a feasible anchor $\widehat{B}_l$, without certifying global optimality (Appendix B.2). We then increase the shared τ, relaxing each layer's limit to $(1+\tau)\widehat{B}_l$. At each limit, we locally refine both new KaHyPar partitions and the previous solution, retaining the feasible candidate with the lowest KM1. Increasing the limit preserves the previous solution's feasibility, so KM1 on the profiling window cannot increase along the sweep.

Mapping partitions to balance GPU load and memory. Per-layer partitions satisfy load limits but do not directly balance expert weight storage across devices. We process layers in order, assigning partitions with more experts to logical nodes with less accumulated weight storage. Within each node, load-aware packing and the same cumulative-storage balancing distribute experts across its GPUs (Figure 2c). The logical layout must fit each GPU's memory budget after runtime reservations. Details and costs appear in Appendices B.2 and B.3.

4.3 Online Expert Migration

We plan layout updates in the background and apply accepted layouts while inference continues. The process has four steps: triggering placement search, mapping candidates to physical devices, selecting and confirming a target, and executing weight transfers. Appendix Figure 7 illustrates the physical mapping, transfer rounds, and safe handoff.

Step 1: triggering placement search. The controller tracks the distribution of complete co-activation sets in successive windows. A persistent shift triggers hypergraph placement search on a recent observation snapshot; sample, threshold, and cooldown rules are given in Appendix C.4.

Step 2: mapping candidates to physical devices. For each candidate, a permutation π maps target node j to physical node $\pi(j)$. The two node policies below plan transfers without moving weights; Appendix C.1 gives the device-level mapping.

Min-volume. Let O_{ij} be the weight bytes shared by old physical node i and target node j across all layers, and S the cluster's total expert-weight bytes. Matching these nodes retains O_{ij} bytes locally, giving inter-node migration volume

$$V(\pi) = S - \sum_{j=1}^{M} O_{\pi(j),j}. \tag{8}$$

Maximum-weight perfect matching (Hungarian algorithm (Kuhn, 1955)) maximizes retained bytes and minimizes inter-node migration volume for the fixed target layout.

Min-time. Concentrated endpoint traffic can make a lower-volume mapping slower. Min-time minimizes the worst endpoint transfer-time bound. Let S_i^{old} and S_j^{tar} denote weight bytes on physical node i and in target node allocation j, and $B_i^{\text{out}}, B_i^{\text{in}}$ physical node i's send and receive bandwidths. Assigning j to i retains O_{ij} bytes, requiring $S_i^{\text{old}} - O_{ij}$ bytes sent and $S_j^{\text{tar}} - O_{ij}$ received. With independent send and receive resources, define

$$c_{ij} = \max\left\{\frac{S_i^{\text{old}} - O_{ij}}{B_i^{\text{out}}}, \frac{S_j^{\text{tar}} - O_{ij}}{B_i^{\text{in}}}\right\}, \qquad T_{\text{LB}}(\pi) = \max_j c_{\pi(j),j}. \tag{9}$$

We binary-search the sorted distinct c_{ij} values. For each trial threshold t, we test whether edges with $c_{ij} \le t$ admit a perfect matching between target and physical nodes. The smallest feasible threshold is $t^\star = \min_\pi T_{\text{LB}}(\pi)$. On the edges permitted by $t^\star$, maximum-weight perfect matching with weights O_{ij} selects the minimum-volume mapping attaining this bound (Equation (8)). Actual migration time also depends on memory constraints and serving interference (Appendix C.1).

Step 3: selecting and confirming a target layout. After mapping each candidate to physical devices, the controller compares estimated MoE execution time under the new and current layouts on recent observation windows, recomputing request homes for each layout. Only a feasible candidate exceeding the required gain enters migration, avoiding unproductive updates. Appendix C.4 gives the scoring, confirmation window, gain threshold, and cooldown rules.

Step 4: scheduling and executing weight transfers. For the accepted target, we organize inter-node transfers into the fewest parallel rounds allowed by the endpoint constraints. Using its fixed device mapping, we schedule complete expert transfers across all layers. Let d_i^{out} and d_i^{in} count the experts node i must send and receive. Each round allows at most one outgoing and one incoming expert per node, so the required number of rounds R satisfies

$$R \ge \Delta = \max_i \{d_i^{\text{out}}, d_i^{\text{in}}\}. \tag{10}$$

We attain this bound by representing transfers as a bipartite multigraph, with separate sender and receiver vertices for each node and one edge per expert transfer. Edge coloring decomposes the graph into Δ matchings, each executed as one parallel round. This achieves the minimum round count under these endpoint constraints; Appendix C.2 gives the proof.

Existing expert copies continue serving during each round. Once its transfers finish, all ranks switch the transferred experts to their new locations at a safe MoE boundary. Old copies are released after outstanding uses finish, before the next round starts. This bounds temporary duplication to one incoming expert per node; intermediate layouts and transfer buffers must also fit GPU memory budgets. Request homes and KV caches remain fixed. Appendix C.3 details transfers and handoff.

5 Evaluation

We evaluate home selection, placement, and migration on one eight-H100 server partitioned into four two-GPU logical nodes. Inter-node traffic uses InfiniBand NICs with direct NVLink/P2P disabled; traffic within each logical node uses NVLink, emulating the RDMA path of a multi-node deployment.

5.1 Experimental Setup

Baselines. We disable replicas in all baselines to isolate placement from replica counts, extra weight memory, and replica-aware routing. EPLB tests load balance; GRACE-MoE tests pairwise co-activation grouping; CAP likewise uses expert-pair statistics. LPT and graph cut are controls. Sem-MoE's token-ID scheduling and Director's future-route predictions, fixed source GPUs, and migration would confound a placement-only comparison.

Models and platform. We test OLMoE-1B-7B-0924, Qwen1.5-MoE-A2.7B, and DeepSeek-V2-Lite (Muennighoff et al., 2025; Qwen Team, 2024; DeepSeek-AI, 2024) on held-out Code requests and Code–Math shifts, fixing predictor checkpoints and layouts before testing. Mapping, migration, and adaptation share the four-node setup above: 400 Gb/s RDMA ports between node leaders and NVLink within nodes. Original routers, dense and shared components, and weights are preserved.

Measurements. Paired policies share the backend, requests, admission limits, and memory budgets. Complete prefill time includes prediction, home selection, input movement, model execution, and pipeline drain. Count-balanced homes equalize active sequence counts without expert routes. Appendices D.1 and D.4.1 give the full protocols.

5.2 Pre-Router: Prediction Quality and Overhead

Prediction quality and cost. On source-file-disjoint Code requests, validation recall, full prediction time, and parameter count select shared projection and head widths $d = h = 256$ for all three models (Table 1); each is within one point of the best test recall in the width sweep. Complete prediction takes under one millisecond per 512-token prompt. At native Top-k, Pre-Router's recall exceeds a matched reconstruction of Sem-MoE's token-ID frequency predictor (Li et al., 2026) by 11.44, 17.82, and 15.00 percentage points on OLMoE, Qwen, and DeepSeek, respectively. Appendix D.2.1 details the baseline and predictor selection protocols.

Model	k	Sem-MoE-style	Pre-Router					
		Recall@k (%)	Recall@k (%)	H	d	h	Trainable (M)	Prediction (ms)
OLMoE	8	68.47	**79.91**	2048	256	256	1.84	0.70
Qwen1.5-MoE	4	48.34	**66.15**	2048	256	256	2.47	0.79
DeepSeek-V2-Lite	6	54.97	**69.97**	2048	256	256	2.66	0.87

Table 1: Selected Pre-Router predictors and a reproduced Sem-MoE-style token-ID baseline. For Pre-Router, H, d, and h denote input, projection, and head widths; trainable counts exclude the frozen frontend. Recall targets routed experts at native Top-k on the same 600 test requests; Pre-Router averages three seeds. Its prediction time covers token transfer, frontend, heads, and Top-k for one 512-token prompt on H100.

From predictions to home decisions. On fixed load-balanced layouts, Pre-Router and an actual-route control share the home-selection rule and capacity limits. Both are scored on actual routes; the control also sees them when choosing homes but is not a global optimum. Table 2 reports OLMoE sends and savings versus count-balanced homes. Pre-Router achieves 83.3 % of the send savings obtained by the actual-route control. This fixed-layout comparison isolates home selection; Section 5.3 tests complete prefill time. Appendix Table 6 gives Qwen and DeepSeek controls.

Home selection	Sends (M) ↓	Saved (k) ↑	Actual-route savings (%) ↑
Count-balanced	22.2558	0.00	0.0
Length-aware	22.2122	43.60	2.0
Pre-Router	**20.4110**	**1844.73**	**83.3**
Using actual expert routes	20.0421	2213.68	100.0

Table 2: OLMoE inter-node dispatch on 600 requests with fixed layouts. One send is a useful token payload to another logical node; M/k mean millions/thousands. Saved is relative to count-balanced homes; the last column divides Saved by actual-route savings. All rows use the same capacity limits.

5.3 Expert Placement and Its Interaction with Pre-Router

Communication–load trade-off. All layouts use the same OLMoE training profile. MoEAtlas performs the load-anchor search and shared-τ sweep (Section 4.2); Figure 3 compares EPLB, uncapped LPT, pairwise graph cut, and GRACE-MoE grouping (Han et al., 2025). MoEAtlas, graph cut, and GRACE average three solver seeds; EPLB and LPT are deterministic. Normalized node load sums per-layer peaks relative to ideal loads; KM1 counts extra node visits per token-layer pair.

At $\tau = 0$, normalized peak node load is 1.00055, versus 1.00132 for uncapped LPT, while KM1 is 13.64% below EPLB. At $\tau = 0.1$, KM1 is 26.70% lower than EPLB, with normalized node and GPU peaks of 1.097 and 1.150. At $\tau = 1.5$, both KM1 and node peak are below GRACE-MoE grouping, by 11.95% and 5.92%. Appendix D.3.1 gives all slacks and resident weight counts.

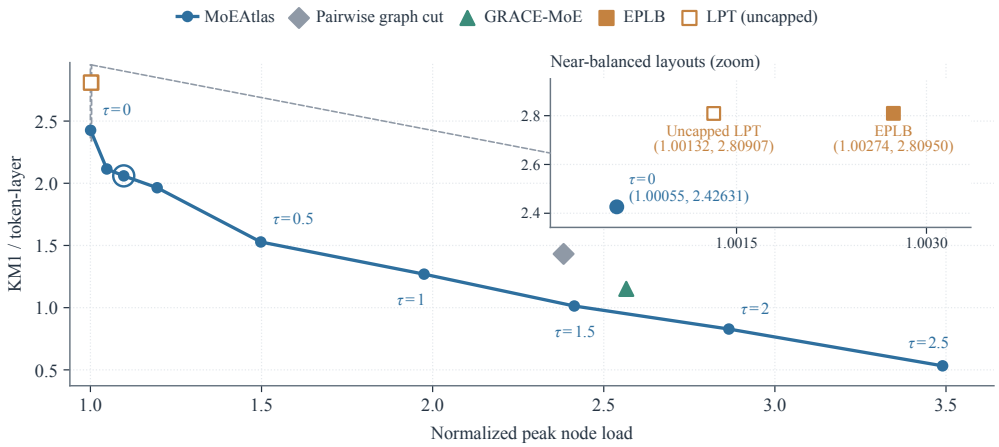

Figure 3: OLMoE communication–load trade-off on the profiling window. MoEAtlas varies shared τ; the ring marks $\tau = 0.1$. The inset zooms into near balance. Lower is better on both axes.

Performance on new requests. With $\tau = 0.1$ fixed before timing, hypergraph placement with count-balanced homes (HG+CB) reduces complete prefill time on 512 new requests per model (64 per batch, three paired seeds) by 17.42 %, 14.94 %, and 22.71 % versus EPLB+CB on OLMoE, Qwen, and DeepSeek (Table 3). A paired OLMoE control finds 9.07 % lower time than GRACE grouping without replication or replica routing. HG+PR adds 2.31–2.57 % and is fastest across all three models.

Model	Complete prefill (s)				MoEAtlas reduction (%)	
	EPLB+CB	GRACE+CB†	HG+CB	**HG+PR**	vs EPLB+CB	vs GRACE+CB
OLMoE	5.024	4.512	4.148	**4.052**	19.33	10.19
Qwen	3.165	2.99	2.692	**2.627**	17.00	12.14
DeepSeek	4.178	3.59	3.230	**3.147**	24.69	12.35

Table 3: Complete prefill time (s) on 512 new requests per model. EPLB: load-balanced placement; HG: hypergraph placement; CB: count-balanced homes; PR: Pre-Router homes. MoEAtlas is HG+PR; its EPLB reductions average paired percentages. †GRACE includes grouping only, without replication or replica routing. The GRACE reduction compares mean prefill times.

5.4 Online Expert Migration

Mapping and transfer scheduling. Across 21 paired OLMoE transitions, min-volume mapping cuts inter-node weight bytes by 11.95 % and transfer time by 10.47 % versus fixed labels; min-time cuts time by 11.82 % (Matching rounds throughout). With min-volume mapping fixed, Matching cuts time by 64.06 % versus Serial and 18.75 % versus Greedy. Appendix D.5.1 reports endpoint pressure and paired variation (Figure 6).

Migration during inference. We time 512 OLMoE requests across three paired layout seeds with one target and transfer plan. Blocking moves weights before serving; online variants serve during InfiniBand transfers. Only pacing caps transfer at 10 GB/s per node and direction.

Policy	Rate cap (GB/s)	Migration (s)	Batch interval (ms)	Complete (s)
Target ready	—	—	128.1	2.051
Blocking	No cap	0.938	128.0	3.004
Online, unrestricted	**No cap**	**2.541**	**173.9**	**2.799**
Online, paced	10	2.770	169.0	2.829

Table 4: OLMoE migration during inference. Complete time includes 512 prefills and transfer drain; batch interval averages 15 completion gaps. Values average three layout-seed medians.

Both online variants finish sooner than blocking: paced saves 5.86 %. Pacing shortens the batch interval versus unrestricted transfer, but prolongs migration and leaves a final drain, so unrestricted finishes first (Table 4). Both intervals remain above the target-ready reference. Peak extra memory is 240 MiB per GPU. Appendix D.5.2 details pacing, round-wise handoff, and timings.

5.5 End-to-End Online Adaptation

On a 30-minute Code–Math–Code stream, we compare frozen, 512 s periodic, and drift-triggered layouts. Updates share the complete shared-τ search, candidate checks, and uncapped InfiniBand transfers while serving continues.

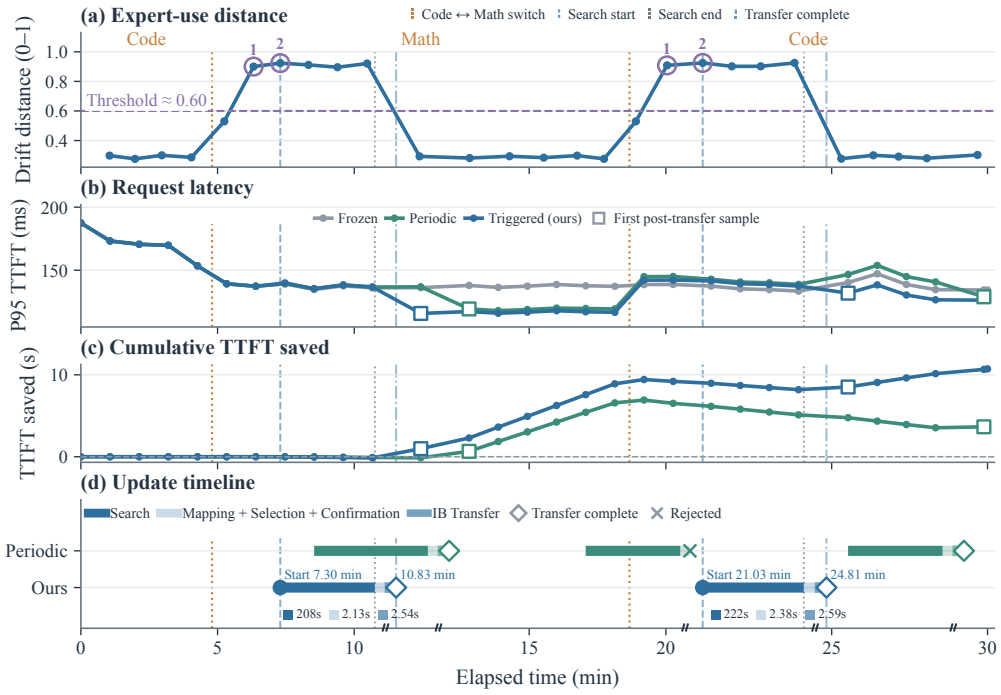

Figure 4: End-to-end Code–Math–Code adaptation over 30 minutes. (a) Expert-use distance; (b) P95 TTFT; (c) cumulative P95-based savings versus frozen; (d) search, confirmation, and transfer. Event guides mark workload changes and layout updates; open squares mark first post-transfer samples.

Figure 4 shows triggered searches after both workload changes. Triggered migration installs the Math layout before the first periodic update and later restores a Code layout sooner. Despite the background search cost, two triggered searches yield greater cumulative P95-based TTFT savings at 30 minutes than the three periodic searches. Appendix D.5.3 gives detector and event timings.

6 Conclusion

MoEAtlas links request-level expert forecasts, complete-set communication costs, and observed workload drift. These signals guide request homes, constrained hypergraph placement, and gradual deployment while serving continues. Paired experiments connect the three modules to complete inference and workload adaptation.

AI Use Statement

We used generative AI to polish manuscript language, find related work, and support research ideation and execution (algorithm design, experimental planning, code assistance, mathematical checks, and figure preparation). The authors reviewed the outputs and take responsibility for the paper.

Reproducibility Statement

Appendix C.2 proves the migration-round bound. Appendices A, B, and C detail Pre-Router, placement, and migration. Appendix D records data separation, baselines, timing boundaries, and statistical aggregation. Experiment records retain source snapshots, input and checkpoint hashes, configuration manifests, routing summaries, and raw repeated measurements. The figure and table scripts read these records directly.

Ethics Statement

MoEAtlas changes physical execution while retaining the target model's router and expert computations. Workloads and model checkpoints are used under their respective access and licensing terms. Released artifacts contain aggregate routing statistics or authorized request identifiers. Numerical consistency and continuation quality are evaluated alongside efficiency.

References

Yaroslav Akhremtsev, Tobias Heuer, Peter Sanders, and Sebastian Schlag. Engineering a direct k-way hypergraph partitioning algorithm. In *Proceedings of the Nineteenth Workshop on Algorithm Engineering and Experiments*, pp. 28–42, 2017. doi: 10.1137/1.9781611974768.3. URL https://epubs.siam.org/doi/10.1137/1.9781611974768.3.

Abhimanyu Bambhaniya, Geonhwa Jeong, Jason Park, Jiecao Yu, Jaewon Lee, Pengchao Wang, Changkyu Kim, Chunqiang Tang, and Tushar Krishna. Scaling multi-node mixture-of-experts inference using expert activation patterns, 2026. URL https://arxiv.org/abs/2604.23150v1.

Sangjin Choi, Sukmin Cho, Yifan Xiong, Ziyue Yang, Youngjin Kwon, and Peng Cheng. ELDR: Expert-locality-aware decode routing for PD-disaggregated MoE serving, 2026. URL https://arxiv.org/abs/2607.00466v1.

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

DeepSeek-AI. Day 6: One more thing, DeepSeek-V3/R1 inference system overview, 2025a. URL https://github.com/deepseek-ai/open-infra-index/blob/main/202502OpenSourceWeek/day_6_one_more_thing_deepseekV3R1_inference_system_overview.md. Open Source Week. Accessed September 17, 2026.

DeepSeek-AI. Expert parallelism load balancer, 2025b. URL https://github.com/deepseek-ai/EPLB. Accessed September 12, 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://jmlr.org/papers/v23/21-0998.html.

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

R. L. Graham. Bounds on multiprocessing timing anomalies. *SIAM Journal on Applied Mathematics*, 17(2):416–429, 1969. doi: 10.1137/0117039. URL https://epubs.siam.org/doi/10.1137/0117039.

Yu Han, Lehan Pan, Jie Peng, Ziyang Tao, Hanqi Zhu, Wuyang Zhang, and Yanyong Zhang. GRACE-MoE: Grouping and replication with locality-aware routing for efficient distributed MoE inference, 2025. URL https://arxiv.org/abs/2509.25041v4. Version 4, May 2026.

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.

Brian W. Kernighan and Shen Lin. An efficient heuristic procedure for partitioning graphs. *The Bell System Technical Journal*, 49(2):291–307, 1970. doi: 10.1002/j.1538-7305.1970.tb01770.x.

Harold W. Kuhn. The Hungarian method for the assignment problem. *Naval Research Logistics Quarterly*, 2(1–2):83–97, 1955. doi: 10.1002/nav.3800020109.

Jiamin Li, Yimin Jiang, Yibo Zhu, Cong Wang, and Hong Xu. Accelerating distributed MoE training and inference with Lina. In *2023 USENIX Annual Technical Conference*, pp. 945–959, 2023. URL https://www.usenix.org/conference/atc23/presentation/li-jiamin.

Yan Li, Zhenyu Zhang, Zhengang Wang, Pengfei Chen, and Pengfei Zheng. Semantic parallelism: Redefining efficient MoE inference via model-data co-scheduling. In *International Conference on Learning Representations*, 2026. URL https://arxiv.org/abs/2503.04398v5.

Qianli Liu, Kaibin Guo, Zicong Hong, Peng Li, Fahao Chen, Haodong Wang, Jian Lin, and Song Guo. Director: Accelerating distributed MoE serving via online proactive expert placement, 2026. URL https://arxiv.org/abs/2607.08782v1.

Shuqing Luo, Pingzhi Li, Jie Peng, Yang Zhao, Yu Cao, Yu Cheng, and Tianlong Chen. Occult: Optimizing collaborative communications across experts for accelerated parallel MoE training and inference. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 41235–41253. PMLR, 2025. URL https://proceedings.mlr.press/v267/luo25f.html.

Niklas Muennighoff, Luca Soldaini, Dirk Groeneveld, Kyle Lo, Jacob Morrison, Sewon Min, Weijia Shi, Pete Walsh, Oyvind Tafjord, Nathan Lambert, Yuling Gu, Shane Arora, Akshita Bhagia, Dustin Schwenk, David Wadden, Alexander Wettig, Binyuan Hui, Tim Dettmers, Douwe Kiela, Ali Farhadi, Noah A. Smith, Pang Wei Koh, Amanpreet Singh, and Hannaneh Hajishirzi. OLMoE: Open mixture-of-experts language models. In *International Conference on Learning Representations*, 2025. URL https://arxiv.org/abs/2409.02060.

Qwen Team. Qwen1.5-MoE: Matching 7B model performance with 1/3 activated parameters, 2024. URL https://qwenlm.github.io/blog/qwen-moe/.

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, 2022. URL https://proceedings.mlr.press/v162/rajbhandari22a.html.

Noam Shazeer, Azalia Mirhoseini, Krzysztof Maziarz, Andy Davis, Quoc Le, Geoffrey Hinton, and Jeff Dean. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. In *International Conference on Learning Representations*, 2017. URL https://arxiv.org/abs/1701.06538.

Xiao Shi, Yingying Sun, Jiangsu Du, Zhiguang Chen, and Yutong Lu. Communication-aware placement and pruning for efficient mixture-of-experts inference, 2026. URL https://arxiv.org/abs/2607.05116v1.

A PRE-ROUTER

We collect predictor design and request-home selection here. Accuracy, runtime, and decision diagnostics are reported in Appendix D.2.

A.1 PREDICTOR ARCHITECTURE AND TRAINING

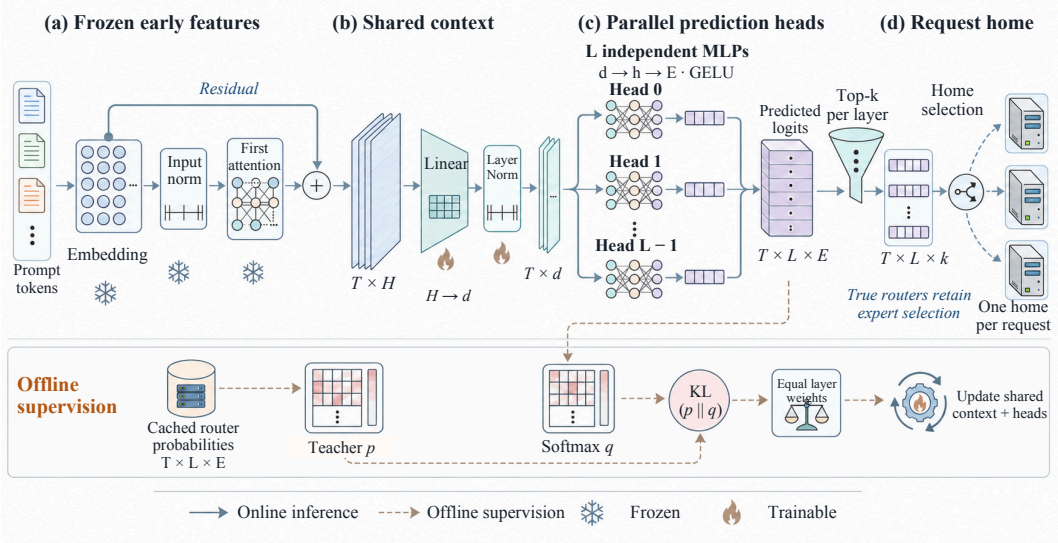

Figure 5: Pre-Router with parallel prediction heads. Frozen early features feed a shared projection and independent layer heads; their predictions guide home selection under the current layout.

Parallel prediction heads. Each layer head uses Equation (5) with $f_l(z) = V_l\,\mathrm{GELU}(U_l z + b_l) + c_l$. All heads read the same shared projection z_t and produce their predictions independently. The capacity study varies projection width d and head width h; the system configuration uses the validation-selected $d = h = 256$ architecture.

OLMoE training and evaluation. The source-file-separated OLMoE capacity study follows Appendix D.2.1. OLMoE system experiments use the following fixed request-level partition. The 4,000-request code dataset is split into 2,800 training-pool requests, 600 test requests, and 600 reserved evaluation requests. Fold 0 of a five-fold partition of the training pool, with split seed 1234, supplies 2,240 training and 560 validation requests. The test set contains 536,199 tokens. We hold this partition fixed across initialization seeds 1234, 1235, and 1236. No identical token sequence occurs in more than one of the training, validation, and test sets. The predictor trains for five epochs using Adam with learning rate 10^{-3}, one unpadded request per update, and equal KL weights on all 16 layers. The teacher targets are the original router probabilities, renormalized after FP16 storage. Minimum token-weighted validation KL selects the checkpoint; test recall averages $|\widehat{A}_{t,l} \cap A_{t,l}|/8$ over tokens and layers. Frozen frontend activations are cached for offline training. Every inference timing reported in Appendix D.2 recomputes the frontend from token IDs.

A.2 REQUEST-HOME SELECTION AND ADMISSION

Pre-Router processes requests in arrival order, updating traffic and sequence counts after each assignment. It selects homes using the communication-time score in Equation (13). Ties favor GPUs with fewer newly assigned requests. For an incoming batch $\mathcal{B}$, each GPU's remaining sequence capacity K_g^{seq} constrains home selection:

$$\sum_{s \in \mathcal{B}} \mathbf{1}[c_s = g] \le K_g^{\mathrm{seq}}. \tag{11}$$

For fixed prefill batches, $K_g^{\mathrm{seq}} = \lfloor \gamma |\mathcal{B}|/P \rfloor$; continuous serving subtracts active decoding sequences from the per-GPU limit. Whole requests use the same sequence limits under every policy. Requests remain queued until a home has a free slot; maximum prompt and output lengths bound the corresponding attention and KV reservations. Evaluation includes batch formation, ingress processing, statistics transfer, home selection, input movement, and synchronization on the critical path. These admission limits control attention and KV demand; placement constrains expert load separately (Appendix B).

Endpoint communication-time estimator. For batch routes A, layout x, and homes $\mathbf{c}$, we aggregate bytes sharing a send or receive endpoint within each communication stage. With independent send and receive service, the stage's bandwidth component is

$$\widehat{t}_{\mathrm{bw}} = \max_r \left\{ \frac{V_r^{\mathrm{out}}}{B_r^{\mathrm{out}}}, \quad \frac{V_r^{\mathrm{in}}}{B_r^{\mathrm{in}}} \right\}, \tag{12}$$

where V_r is endpoint traffic in bytes and B_r its directional bandwidth. After accounting for stage service costs, the layer's dispatch and combine estimates sum to

$$\widehat{T}_{\mathrm{comm}}(x, \mathbf{c}; A) = \sum_{l=1}^{L} \big(\widehat{t}_l^{\,\mathrm{disp}}(x, \mathbf{c}; A) + \widehat{t}_l^{\,\mathrm{comb}}(x, \mathbf{c}; A) \big). \tag{13}$$

Dispatch, expert computation, and combine are sequential within each layer; expert computation runs in parallel across GPUs. For fixed routes and layout, expert computation does not depend on the request homes. Equation (3) therefore uses this communication estimate for both home and layout scoring, adding the peak GPU expert time only when ranking layouts.

B Hypergraph Expert Placement

We first verify why complete activation sets are needed to count node visits, then detail candidate generation and its computational cost. Placement experiments and measured search costs appear in Appendix D.3.

B.1 Pairwise statistics and node-coalesced traffic

Six-expert example in Figure 2a. Using expert IDs $1, \ldots, 6$, let the two activation records be

$$\mathcal{H}_{\mathrm{A}} = \{\{1,2,3\}, \{1,4,5\}, \{2,4,6\}, \{3,5,6\}\},$$
$$\mathcal{H}_{\mathrm{B}} = \{\{1,2,4\}, \{1,3,5\}, \{2,3,6\}, \{4,5,6\}\}.$$

Each set occurs once. In both records, every expert occurs twice, and every unordered pair occurs once except $(1,6)$, $(2,5)$, and $(3,4)$, which never occur together. Both therefore induce the same twelve-edge graph with unit edge weights. The complete sets differ: for example, $\{1,2,3\}$ appears in A, whereas its three pairs appear in three separate tokens in B.

Fix Node 1 to $\{1,2,3\}$, Node 2 to $\{4,5\}$, and Node 3 to $\{6\}$. Expert-call loads are $(6,4,2)$ for both records. The token-wise node counts are $(1,2,3,3)$ for A and $(2,2,2,2)$ for B, giving mean KM1 costs $(0+1+2+2)/4 = 1.25$ and $(1+1+1+1)/4 = 1$. For a fixed request home on Node 2, the useful dispatch counts are $(1,1,2,2)$ and $(1,1,2,1)$: six versus five remote payloads in total. Thus the information lost by pairwise aggregation changes both node coverage and communication under the same expert layout and request home.

Director's per-source counts. Place all four activations of each six-expert record above on source GPU j in Node 2. Every expert occurs twice, so Director's per-source counts are $W_{j,i}^l = 2$ for $i = 1, \ldots, 6$ in both records (Liu et al., 2026); source–destination totals derived from them also coincide. Under the same layout and home, node-coalesced dispatch still needs six versus five remote payloads.

Sem-MoE's token-ID frequencies. Form eight Top-4 activations by choosing one expert from each of four pairs $(e_{i,0}, e_{i,1})$. Let eight occurrences of one token ID use all even-parity selections in profile

E and all odd-parity selections in profile O. Each expert appears four times in both profiles, and their pairwise counts also coincide. With all other data fixed, Sem-MoE's token-ID–expert frequencies and co-clustering inputs are identical (Li et al., 2026). Place first members on Node 0 and second members on Node 1, with home Node 0. E has two one-node and six two-node selections; O has eight two-node selections. Both profiles have equal expert loads and four experts per node, but require seven and eight remote payloads, respectively. This is a distinction in node-coalesced traffic, not a measured comparison with Sem-MoE's per-expert communication path.

Pairwise cut versus node traffic. One token selects $\{A, B, C, D\}$, inducing six unit-weight pairs. With home Node 0, compare the budget-feasible placements $\{A, B\}|\{C, D\}$ and $\{A\}|\{B, C, D\}$ across Nodes 0 and 1. The first cuts AC, AD, BC, BD (weight 4); the second cuts AB, AC, AD (weight 3). Both have KM1 cost 1, one remote dispatch payload, and one combine payload. Pairwise cut favors the second layout although node-coalesced traffic is unchanged.

B.2 Demand estimation and placement search

At update j, actual routes in the profiling window give one complete selected-expert set per token and layer. For N_j prompt tokens, its normalized frequency $p_{l,a}^{(j)}$ becomes hyperedge count $w_{l,a}^{(j)} = N_j p_{l,a}^{(j)}$; expert load is $C_{l,e}^{(j)} = \sum_{a:e\in a} w_{l,a}^{(j)}$. Layer-wise normalization separates routing composition from request volume. The profile retains expert IDs, counts, request boundaries, and timestamps for graph construction and candidate evaluation. Appendix C.4 specifies window and sampling rules. Predictor weights remain fixed during serving; placement changes remap expert IDs to nodes.

Algorithm 1 constructs complete candidate layouts. For each layer, Equation (7) gives a load lower bound; uncapped LPT supplies a feasible upper bound. KaHyPar probes the interval by bisection, and the smallest feasible upper bound found becomes $\widehat{B}_l$. This is a feasible heuristic anchor, not a certificate of global load optimality: failure across the tried seeds does not prove the tested cap infeasible. Each layer has its own anchor; the shared ascending τ relaxes all layer limits together. At each slack, partitions across layers form one layout per seed, followed by GPU-load and weight-storage balancing. Thus S slacks yield at most S candidates per seed, without enumerating per-layer slack combinations. Confirmation follows Appendix C.4.

Algorithm 1 Construct resource-feasible candidate layouts

Input: snapshot $w^{(j)}$; shared load slacks; device budgets; tolerance δ.
Output: a set $\mathcal{X}$ of feasible logical layouts spanning all layers.

```
Build layer-wise hyperedges; compute C_{l,e} from w^(j).
For each layer l independently:
    Set lo = B_l^LB (Equation (7)); an LPT placement b_l with no count cap supplies its peak node load as hi.
    While hi − lo > δ: test v = (lo + hi)/2 with KaHyPar.
        If a partition with peak load ≤ v is found, store it and set hi = v; otherwise set lo = v.
    Set B̂_l = hi and refine stored b_l under this bound.
For each shared τ in ascending order:
    For each layer, refine the previous b_l and new KaHyPar partitions under (1 + τ)B̂_l; retain the lowest KM1.
    Balance cumulative weight storage by layer-wise node permutations; pack experts within nodes and balance GPU labels.
    Add the resulting layout to 𝒳 if it fits device memory; retain its GPU-load estimates for candidate evaluation.
Return 𝒳 for evaluation with Pre-Router and transition-memory checks.
```

Partitioning and refinement. KaHyPar (Akhremtsev et al., 2017) uses the direct KM1 objective, expert-call vertex weights, and activation-frequency hyperedge weights. We check its node loads; if infeasible, we attempt LPT packing under the same limit. Only a feasible candidate enters local refinement on the original hypergraph. Each round applies the feasible expert move or cross-node swap with the largest strictly positive KM1 reduction. A simultaneous swap can satisfy the limit even when either move alone cannot. Each call permits at most four move rounds followed by two swap

rounds, stopping a stage when no improvement remains. At every slack, we also refine the previous feasible solution.

Settings, mapping, and load accounting. The complete search uses three independent seeds, all training hyperedges, a bisection tolerance of 128 expert calls, and no per-layer wall-clock cutoff. Larger node groups go to nodes with less accumulated weight storage; within-node LPT and GPU-label permutations balance computation and GPU storage. Expert weights have equal size within each evaluated model, so counts measure storage. These permutations preserve node KM1 and peak node load; final checks include model weights and serving workspaces. The online CPU worker runs the same search on its observed snapshot. Appendices D.3.1 and D.5.1 give offline and migration-layout protocols, respectively.

For each layer, node partitions are sorted by decreasing expert count and assigned to logical nodes in increasing accumulated weight storage; the storage totals are updated before the next layer. This relabeling preserves KM1 and peak node load. Within a node, experts in each layer are packed by decreasing call load onto the currently least-loaded GPU. GPU assignments are then relabeled so larger expert counts occupy GPUs with less accumulated weight storage. This preserves the layer's peak GPU load, and final layouts must fit each GPU's memory budget after runtime reservations.

Normalized node load is $\sum_l \max_m C_{l,m}$ divided by $\sum_{l,e} C_{l,e}/M$, where $C_{l,m} = \sum_{e:x_l(e)=m} C_{l,e}$; one is ideal balance. GPU load is analogous after packing: GPU g receives $\sum_{e:g_l(e)=g} C_{l,e}$ calls in layer l. Each selected expert counts once even when experts share a communication payload. Both load metrics use actual routes in the evaluated workload. Home admission separately limits attention and KV demand. For mixed prompt lengths, we record actual token counts and peak device allocation under those limits.

B.3 Computational cost of placement search

For N observed tokens, E experts, and Top-k routing, let H_l be the distinct selected sets in layer l and I_l their total incidences. Then

$$H_l \le \min\{N, \tbinom{E}{k}\}, \qquad I_l = kH_l. \tag{14}$$

Repeated sets increase edge weights without adding edges. Sorting and aggregating the observed sets takes $O(Nk \log k + Nk \log N)$ per layer and $O(E + I_l)$ graph storage, besides the routing input and sorting workspace.

With initial LPT upper bound B_l^{UB}, lower bound B_l^{LB}, and bisection tolerance δ, at most $J_l = \lceil \log_2 \max\{1, (B_l^{\mathrm{UB}} - B_l^{\mathrm{LB}})/\delta\} \rceil$ anchor probes are needed. For S shared slack values, one seed trajectory uses at most $\sum_l (J_l + S)$ KaHyPar solves and $1 + 2S$ additional refinement calls per layer. Each solve includes coarsening, initial partitioning, and internal refinement; its duration depends on the observed graph and requested load limit.

With cached expert incidences, a complete move scan costs $O(MI_l + EM^2)$ and a swap scan costs $O(EI_l + E^2M)$, including load checks. Each additional refinement permits at most four moves and two swaps. The EI_l swap term can be substantial when $E \gg M$; measured graph sizes and complete-search elapsed time appear in Appendix D.3.3.

C Online Expert Migration

This appendix gives the matching guarantees, transfer-round construction, memory handoff rules, and controller for the update process in Section 4.3. Matching fixes the physical layout before transfers begin; only the accepted target is applied.

C.1 Whole-model migration matching

Let $s_{l,e}$ be expert weight size, $x_l^{\mathrm{old}}(e)$ its physical source node, and $y_l(e)$ its logical target node. Across all layers, define

$$O_{ij} = \sum_{l,e:\, x_l^{\mathrm{old}}(e)=i,\, y_l(e)=j} s_{l,e}. \tag{15}$$

One node permutation π maps complete target allocations across all layers to physical nodes, preserving each layer's connectivity and load. Min-volume maximizes $\sum_j O_{\pi(j),j}$ by maximum-weight matching. For min-time, the smallest feasible threshold on c_{ij} in Equation (9) is found by testing perfect matchings; a maximum-overlap matching among its permitted pairs breaks ties toward less transferred weight. The same overlap rule maps complete target GPU allocations within each matched node. These optima apply to the fixed target layout with interchangeable nodes and GPUs; the resulting mapping still passes memory and execution-time checks.

C.2 Endpoint bounds and transfer rounds

Represent complete inter-node expert moves as edges between sender and receiver copies of the nodes. Each round is a matching, so at least $\Delta = \max_i\{d_i^{\text{out}}, d_i^{\text{in}}\}$ rounds are required. Adding dummy edges makes the graph Δ-regular; Hall's theorem gives a perfect matching, and removing one repeatedly yields exactly Δ rounds after dummy edges are discarded. With equal expert size s and common full-duplex endpoint bandwidth B, the endpoint bound is $T_{\text{LB}} = s\Delta/B$.

C.3 Memory-feasible migration and version handoff

Only the accepted target is transferred. Each remote round moves at most one outgoing and one incoming complete expert per node; local GPU moves are scheduled separately. Final and intermediate layouts must fit device memory, including incoming copies and transfer buffers. Old copies serve during transfer. After each round, all ranks publish the completed experts' new locations at a common MoE boundary, wait for outstanding uses, and retire old copies before the next round. The fixed-target OLMoE study uses 12 MiB BF16 experts and a 256 MiB/GPU extra-memory allowance. Per-round allocation, remapping, and retirement are timed; reported memory includes incoming copies and transport workspace. Appendix D.5.1 reports the transfers.

C.4 Online control flow and drift confirmation

Let $p_{l,a}^{(j)}$ be normalized co-activation set counts in window j and $p_{l,a}^{\text{ref}}$ the reference from the last completed layout evaluation. The mean total-variation drift score is

$$d_j = \frac{1}{2L} \sum_{l,a} |p_{l,a}^{(j)} - p_{l,a}^{\text{ref}}|. \tag{16}$$

For physical layout x and batches $\mathcal{B}(W)$ in window W, the execution-time score is

$$\widehat{J}(x; W) = \sum_{b \in \mathcal{B}(W)} \widehat{T}_{\text{MoE}}\big(x, \mathbf{c}_b(x); A_b\big), \tag{17}$$

where $\mathbf{c}_b(x)$ is the home assignment recomputed from stored Pre-Router predictions and A_b contains observed expert routes. The minimum relative gain threshold is

$$\widehat{J}(x^{\text{new}}; W_{\text{check}}) < (1 - \theta_g)\widehat{J}(x^{\text{current}}; W_{\text{check}}). \tag{18}$$

The initial reference is the observed profile of the starting layout. Qualified windows meet request, token, and fresh-sample minimums; insufficient windows reset the exceedance counter. The reference remains fixed while monitoring. After r consecutive windows satisfy $d_j > \theta_d$, the controller searches only if no update is active and cooldown has expired.

The search ranks feasible layouts on snapshot W_{fit} using Pre-Router homes and observed expert routes, then selects the lowest-score candidate. Its selected target is checked on fresh, nonoverlapping requests W_{check} for load and memory feasibility and, when the current layout meets its load budgets, the gain in Equation (18). After this check, p^{ref} becomes the W_{check} profile and the counter resets, even if the target is rejected. At most one search or migration is active through final retirement. Each microbatch uses one published expert map; requests retain their home and KV cache while expert locations change.

The long-horizon experiment uses nonoverlapping 64-request windows, $\theta_d = 0.60$, and $r = 2$ (Figure 4). Gain and cooldown thresholds are fixed on validation requests, resampling by request.

D EXPERIMENTAL PROTOCOL AND ADDITIONAL RESULTS

D.1 EXPERIMENTAL SETUP

Models and requests. OLMoE/Qwen/DeepSeek use 16/24/26 MoE layers, 64/60/64 routed experts, Top-8/4/6 selection, hidden width 2,048, and routed-expert intermediate widths 1,024/1,408/1,408. Qwen retains its local width-5,632 shared expert and sigmoid gate; DeepSeek retains its dense first layer, MLA, two shared experts per MoE layer, and 26 prediction heads for decoder layers 1–26. Code requests have fixed IDs and 2,240/560/600 train/validation/test sizes. Capacity and home-decision controls split complete source files (Appendix D.2.1); system runs use request-level fold 0 (seed 1234) of the original 2,800-request training pool. Identical requests never cross splits, although source fragments may. Qwen and DeepSeek retokenize the same IDs. Retained lengths are at most 1,024 tokens for the OLMoE predictor study and 512 for Qwen and DeepSeek; complete prefill uses the first 512 variable-length test requests. Code–mathematics streams have separate unique-request manifests. Experiment hashes identify configurations, checkpoints, layouts, and inputs.

Comparison policies. Timed policies share backend, router, precision, requests, and sequence limits. EPLB balances expert load without replicas; pairwise minimizes weighted cut; GRACE-MoE uses spectral grouping without replication or replica routing. Hypergraph placement uses complete selected sets under load and memory limits; random and hotspot layouts are controls. Home-selection policies share admission limits; actual routes are available only to the offline control. Frozen, periodic, and triggered updates share search, confirmation, and migration and differ in their trigger.

Hardware and timing. Eight H100 80 GB GPUs form four logical two-GPU nodes on one server. Each leader has a 400 Gb/s InfiniBand port; local exchanges use NCCL/NVLink, with NVSHMEM P2P and NVLS shortcuts disabled remotely. The 2 GiB symmetric heap and fixed transport workspaces sit outside PyTorch allocations. Software and NIC inventories are recorded; quoted NVLink peak sums both directions over 18 links/GPU. Communication is calibrated on this runtime. End-to-end runs use BF16 prediction; the capacity study separately reports FP32-head accuracy and latency. Server-side TTFT spans scheduled arrival to the first token ready at home, including queueing, prediction, scheduling, and prefill. Complete prefill uses two warmups, five timed repetitions, three paired layout/assignment seeds (36 conditions), and rotated policy order. Loading is excluded; predictor startup, input movement, and drain are included. Statistics aggregate paired runs, while quality intervals resample requests. Table 3 reports complete prefill.

D.2 PRE-ROUTER EVALUATION

D.2.1 JOINT PROJECTION AND HEAD-WIDTH STUDY

We reuse Python fragments from the local CodeParrot corpus and group them by the SHA-256 of the complete source text. The resulting 2,240/560/600 training, validation, and test requests comprise 463/318/327 disjoint source-document groups. No identical source document or token sequence crosses these splits. Repository identifiers are unavailable, so this is source-file-level isolation. All models share request identities, with at most 1,024 tokens for OLMoE and 512 for Qwen and DeepSeek. Model tokenizers, frozen frontends, and original router supervision remain model-specific.

The grid is $d \in \{64, 128, 256\}$ and $h \in \{128, 256, 512\}$ for each model, with initialization seeds 1234–1236: 81 training conditions in total. We train only the shared linear projection, LayerNorm, and parallel $d \to h \to E$ GELU heads. Adam uses learning rate 10^{-3}, five epochs, and one unpadded request per update. The objective sums equal-weighted KL losses over routed layers against full router probabilities; Top-k weights alone are not the supervision distribution. Minimum validation KL selects the epoch. Cached frozen features accelerate training only.

Before evaluating test requests, we retain configurations within one percentage point of the best mean validation recall, then those within 5% of the fastest eligible full prediction time at $B = 1, T = 512$, and select the fewest trainable parameters. Higher validation recall breaks parameter ties. Table 5 shows the OLMoE trade-off around the validation-selected configuration; all 81 configurations were considered before TEST. The system experiments use the validation-selected $d = h = 256$

architecture on their fixed request-level partition. Table 1 reports recall from the separate source-file-disjoint capacity study.

H	d	h	Trainable (M)	Recall@8 (%)	Prediction (ms)
2048	64	256	0.66	77.88	0.70
2048	128	256	1.05	79.12	0.71
2048	256	128	1.18	79.26	0.69
2048	**256**	**256**	**1.84**	**79.91**	**0.70**
2048	256	512	3.16	80.30	0.73

Table 5: OLMoE capacity trade-off. Rows vary one width while fixing the other at the selected value (bold). Recall averages three training seeds on 600 test requests. Prediction measures one 512-token prompt on H100, including token transfer, the frozen frontend, all heads, and Top-k.

Prediction baseline. The Sem-MoE-style control in Table 1 estimates each token-ID–expert affinity from the same 2,240 source-file-disjoint training requests as Pre-Router. For a test ID absent from training, it uses the nearest seen ID in the original model's embedding space; no validation or test routes enter the lookup. Both methods are evaluated against actual Top-k sets on the same 600 held-out requests. With exactly k predicted and selected experts, set-member recall equals precision and F1.

Isolating Pre-Router home selection. We compare home policies on the same three EPLB expert placements per model and 600 test requests (batches of 64, with 24 in the last batch). Count-balanced homes equalize request counts with deterministic shuffling; length-aware homes balance token counts. Predicted and actual routes enter the same calibrated scheduler. All policies use the per-GPU capacity $\lceil 1.25B/8 \rceil$, with three predictor seeds per placement.

Using observed routes for every policy, we count one useful dispatch payload per visited non-home node, excluding padding, metadata, and network timing. The actual-route offline control can trade payload volume for endpoint time; it is neither a minimum-volume nor a minimum-time oracle. Table 2 gives the OLMoE comparison; Table 6 reports the remaining two models.

Home selection	Sends (M) ↓	Saved (k) ↑	Actual-route savings (%) ↑
Qwen1.5-MoE			
Count-balanced	13.7637	0.00	0.0
Length-aware	13.7679	-4.20	-0.3
Pre-Router	**12.6795**	**1084.23**	**79.2**
Using actual expert routes	12.3947	1368.98	100.0
DeepSeek-V2-Lite			
Count-balanced	18.8148	0.00	0.0
Length-aware	18.8179	-3.10	-0.2
Pre-Router	**17.2035**	**1611.27**	**86.1**
Using actual expert routes	16.9434	1871.39	100.0

Table 6: Qwen and DeepSeek home-selection controls on 600 paired request IDs. Expert placement is shared across policies. One send is a useful token payload to another logical node; M/k mean millions/thousands. Saved is relative to count-balanced homes; the last column divides Saved by actual-route savings. Rows share sequence-capacity limits; all use three layout seeds, and Pre-Router also averages three predictor seeds.

D.2.2 Full prediction cost

Full prediction begins with CPU token IDs and ends after GPU Top-k. Table 7 separates frontend and head costs for the parallel $d = 128, h = 512$ timing control on validation prefixes on one H100. Each shape has three warmups and 21 timed repetitions. Isolated times exclude distributed feature exchange and home selection, which are included in Table 3.

Timed work	1×256	1×1024	8×1024	16×1024
Full prediction	0.90	0.99	3.79	7.02
Frontend only	0.60	0.62	2.00	3.68
Heads and Top-k	0.26	0.41	1.81	3.36

Table 7: Parallel predictor complete and component latency (ms). Column headers give batch size $\times$ prompt length; component rows are timed separately.

Layout	Node error	GPU error	Volume gap	Capped score gap
Random	0.0559	0.1180	0.051%	+0.028%
Hotspot	0.1510	0.1543	0.000%	+0.124%
Hypergraph	0.0902	0.1286	0.021%	+0.070%

Table 8: Forecast-to-decision diagnostics on 512 TEST requests and three layout seeds. Node/GPU errors average the normalized layer-wise maximum visit error. Volume gap compares uncapped predicted and true-route homes; capped score gap compares Pre-Router's home-selection rule under predicted and actual routes. These gaps are calculated scores, not latency measurements.

D.2.3 Destination and home-selection diagnostics

For a fixed layout, substituting another expert on the same GPU can leave communication unchanged; changing a destination node can alter the preferred home. We therefore evaluate expert recall together with visit errors and the communication cost of the resulting assignments. For request s, the layer-wise node-visit error is $\frac{1}{LT_s}\sum_l \max_m |\widehat{n}_{s,l,m} - n_{s,l,m}|$; the analogous GPU-visit errors use analogous per-GPU counts; remote-presence errors count token-layer pairs requiring a destination outside the home node. We also report the aggregate diagnostic $\|\widehat{a}_s - a_s\|_\infty/(T_sL)$. These feature errors and expert recall use a fixed layout and the same test requests.

The native complete-prefill comparison in Table 3 evaluates the resulting assignments with the same per-GPU request limits for every policy.

Let $\mathcal{S}_x(A)$ denote the homes produced by Pre-Router's home-selection rule given routes A and layout x, with identical admission limits. The paired communication-score difference is

$$G = \widehat{T}_{\mathrm{comm}}(x, \mathcal{S}_x(\widehat{A}); A) - \widehat{T}_{\mathrm{comm}}(x, \mathcal{S}_x(A); A). \tag{19}$$

Both assignments are evaluated using actual routes. The true-route run is a matched scheduling control; since both runs use a heuristic, G can have either sign. We evaluate eight 64-request TEST windows for each of nine frozen layouts, using the same ten-request limit per GPU. These offline diagnostics use a CPU reference of this home-selection rule, cached FP32-head predictions, and the frozen native calibration. Table 8 reports the mean relative score gap, alongside visit errors and uncapped volume decisions. Exact capacitated volume assignments additionally verify the prediction-error bound in every window.

Matched expert substitutions. Replacing one selected expert in about one fifth of token-layer pairs gives 97.50% recall in all controls. Replacements on the same GPU leave the communication features and selected homes unchanged; replacing experts on another node changes 12.5–53.1% of homes across the three layouts. Thus equal expert recall need not imply equal home decisions.

D.3 Expert placement evaluation

D.3.1 Frozen resource-constrained placement

The cached OLMoE training-pool routing shard contains 512 requests. Under the fixed five-fold split (fold 0, seed 1234), 95 belong to validation; the remaining 417 form the placement profile. Their lengths range from 64 to 1,024 tokens and sum to 359,553. Figure 3 compares optimization on the same observed routing snapshot. We additionally freeze these layouts and evaluate them on 512 new requests (456,890 tokens), disjoint from the 417 requests used for placement. Table 9 reports the slack sweep on new requests. For the selected $\tau = 0.1$ OLMoE layout, observed and new KM1 are

2.0595 and 2.0537, while normalized node peaks are 1.09737 and 1.10192; the generalization check uses the same frozen mapping. MoEAtlas uses every profile token for both complete hyperedges and expert loads. Each layer's node-load limit is $(1+\tau)\widehat{B}_l$, with $\widehat{B}_l$ found by the bisection procedure in Algorithm 1. Layer-wise expert counts may differ; cumulative storage is balanced afterward. Three independent seeds each follow the full ascending, warm-started trajectory over ten slacks: $\tau \in \{0, 0.05, 0.1, 0.2, 0.5, 1, 1.5, 2, 3, 4\}$. We additionally evaluate $\tau = 2.5$ by continuing each seed's $\tau = 2$ solution with the same load anchor, hypergraph, and refinement settings. The original $\tau = 3, 4$ results retain their original warm-start trajectory.

Independent placement baselines. Pairwise graph cut builds a weighted expert-pair graph from all observed routes. Recursive balanced Kernighan–Lin bisection (Kernighan & Lin, 1970) forms four node groups; subsequent pairwise group refinement reduces the cut while preserving group sizes. We use at most 100 bisection iterations and ten refinement sweeps, followed by count-capped GPU packing. It uses neither KaHyPar nor a τ sweep. GRACE-MoE grouping follows hierarchical spectral clustering (Han et al., 2025): node groups have unrestricted sizes, while within-node GPU groups use controlled non-uniformity $r = 0.15$, the reported configuration. Expert reassignment follows its affinity-based size correction. We evaluate grouping without replication. Both stochastic baselines use seeds 0–2; no configuration is selected on TEST.

EPLB assigns experts in descending observed load to the least-loaded eligible node, then repeats packing within each node. It limits each node to 16 experts and each GPU to eight in each layer. The separate *uncapped LPT* control removes both expert-count limits. All layouts store the same 64 experts per layer without replication. Local allocation differs: Table 9 reports the maximum experts on one GPU in one layer. This layout study compares communication and load. Its node/GPU memory counts are recorded alongside each frozen mapping.

Normalized peak load is the sum of layer-wise peak loads divided by the sum of ideal evenly shared loads, computed within each workload. KM1 uses each token's complete actual Top-8 set. Under a uniformly sampled request home, mean remote dispatch payloads per token-layer pair equal $\frac{3}{4}(\overline{\mathrm{KM1}} + 1)$ for four nodes; measured bytes and elapsed time additionally depend on homes, packet sizes, and execution.

Native prefill configuration. Table 3 evaluates layouts from the complete shared-τ search in Section 4.2, with $\tau = 0.1$ fixed before test timing. Hypergraph expert counts may differ across nodes and GPUs within a layer. Device mapping balances per-layer GPU load and cumulative weight storage, and final layouts must fit GPU memory. All placements are frozen before timing on the same new requests.

Layout	τ	KM1/token-layer	Node peak	GPU peak	Max count
EPLB	—	2.8100	1.0102	1.0628	8
LPT (uncapped)	—	2.8096	1.0109	1.0260	12
Pairwise graph cut	—	1.4375	2.3776	2.4117	8
GRACE-MoE grouping	—	1.1555	2.5611	3.8240	26
Hypergraph	0	2.4241	1.0109	1.0336	21
Hypergraph	0.05	2.1100	1.0556	1.0943	18
Hypergraph	0.1	2.0537	1.1019	1.1536	19
Hypergraph	0.2	1.9614	1.2006	1.2493	29
Hypergraph	0.5	1.5223	1.5062	1.5597	22
Hypergraph	1	1.2682	1.9760	1.9915	23
Hypergraph	1.5	1.0130	2.4127	2.4298	24
Hypergraph	2	0.8300	2.8623	2.8812	26
Hypergraph	2.5	0.5360	3.4852	3.5007	29
Hypergraph	3	0.0092	3.9946	4.0123	33
Hypergraph	4	0.0001	3.9999	4.0173	33

Table 9: Frozen layouts evaluated on 512 new requests. Solver rows average three seeds. Node and GPU columns report normalized peak expert load. Max count is the largest expert count per GPU per layer across the reported seeds.

Native prefill against GRACE grouping. We compare the complete-search $\tau = 0.1$ layout with the GRACE-MoE grouping control fitted on the same 417 OLMoE profiling requests. The control uses hierarchical spectral grouping with $r = 0.15$ and no replication; both layouts use the same 512 new requests, count-balanced homes, batch size 64, and native prefill backend. Across three paired layout seeds, each timed five times after two warmups, mean seed-median prefill time is 4.512 s for GRACE grouping and 4.148 s for hypergraph placement, a mean paired reduction of 9.07 %. This is a comparison with GRACE's grouping component, not its full system with replication and replica-aware routing.

D.3.2 Graph controls on Qwen and DeepSeek

Table 10 extends the placement comparison to Qwen's 24 MoE layers with 60 experts and DeepSeek's 26 MoE layers with 64 routed experts. Every method uses the same first 417 training requests as the corresponding full hypergraph search; mappings are frozen before evaluating the first 512 test requests. Pairwise graph cut uses recursive balanced Kernighan–Lin bisection followed by pairwise partition exchanges. GRACE-MoE grouping uses hierarchical spectral clustering and controlled GPU groups with $r = 0.15$, without replication. We relabel node and GPU groups across layers to balance weight storage for both controls, preserving each layer's group memberships, communication cost, and peak expert load.

At $\tau = 0.1$, Qwen's test KM1 is 1.145 versus 1.168 for pairwise, and its normalized node peak is 1.093 versus 1.115. DeepSeek has nearly identical test KM1 under these two methods (1.404), while hypergraph placement lowers normalized node peak from 1.311 to 1.097. GRACE grouping reduces KM1 further, with larger node and GPU peaks. The maximum layer peak also exposes hotspots hidden by an average across layers. All displayed layouts stay within the 4 GiB routed-weight budget; load budgets need not hold for unconstrained graph-grouping controls or larger slack points. These are layout metrics on independent requests, not inference-time speedups.

Quantity	Recorded value
Observed tokens	359,553
Distinct hyperedges per layer (H_l)	57,274–165,976
Total distinct hyperedges ($\sum_l H_l$)	1,634,813
Total expert–hyperedge incidences ($\sum_l I_l$)	13,078,504
Bisection probes per layer and seed	1–4
Partition calls: bisection / slack search	153 / 528
$\tau = 2.5$ continuation: tasks / CPU workers	48 / 16
$\tau = 2.5$ continuation: search wall time (s)	345.50

Table 11: Observed graph size and search effort for 417 OLMoE requests. Graph counts cover one workload snapshot; partition-call counts cover all 16 layers, three seeds, and eleven evaluated slack values. Wall time refers only to the $\tau = 2.5$ continuation, including its KaHyPar calls and local refinement.

Model	Placement	Fit KM1	Test KM1	Node	GPU	Max node	MiB
Qwen	Load-only	1.790	1.790	1.044	1.056	1.060	3152
Qwen	Pairwise	1.161	1.168	1.115	1.156	1.326	2970
Qwen	GRACE grouping	0.967	0.968	1.734	1.976	2.556	3201
Qwen	Hypergraph $\tau = 0$	1.325	1.330	1.011	1.059	1.026	2986
Qwen	Hypergraph $\tau = 0.1$	1.139	1.145	1.093	1.127	1.117	3020
Qwen	Hypergraph $\tau = 0.5$	0.924	0.930	1.493	1.526	1.518	3052
Qwen	Hypergraph $\tau = 1.5$	0.687	0.688	2.424	2.454	2.521	3118
Qwen	Hypergraph $\tau = 2.5$	0.315	0.312	3.488	3.518	3.514	3102
DeepSeek	Load-only	2.372	2.371	1.035	1.052	1.105	3432
DeepSeek	Pairwise	1.395	1.404	1.311	1.335	1.686	3432
DeepSeek	GRACE grouping	1.312	1.315	1.428	1.842	1.861	3482
DeepSeek	Hypergraph $\tau = 0$	1.625	1.627	1.013	1.033	1.029	3482
DeepSeek	Hypergraph $\tau = 0.1$	1.401	1.404	1.097	1.116	1.132	3482
DeepSeek	Hypergraph $\tau = 0.5$	1.067	1.071	1.498	1.515	1.524	3514
DeepSeek	Hypergraph $\tau = 1.5$	0.825	0.826	2.414	2.432	2.526	3614
DeepSeek	Hypergraph $\tau = 2.5$	0.420	0.420	3.434	3.450	3.510	3646

Table 10: Cross-model placement controls. Fit/Test KM1 counts visited nodes minus one per token-layer pair. Node/GPU summarize normalized per-layer peaks as in Figure 3; Max node is the worst normalized layer peak across seeds. MiB is maximum resident routed-weight storage per GPU. Stochastic methods average three seeds, except maxima; load-only is deterministic.

D.3.3 Recorded search scale

Recorded search scale. Table 11 summarizes the completed OLMoE placement records used in Figure 3. The study retains every observed activation set, with $E = 64$, $k = 8$, $L = 16$, and $M = 4$. The three independent trajectories use $\delta = 128$, $U = 4$, and $V = 2$. The table includes the ten original slack values and the separate $\tau = 2.5$ continuation from saved $\tau = 2$ partitions. Its time entry covers that continuation across all layers and seeds, using 16 CPU workers under the recorded concurrent workload. It measures background search elapsed time; foreground latency is evaluated separately in Section 5.4.

D.4 Combined prediction and placement

The three-model comparison in Section 5.3 uses one matched protocol; Appendix D.4.1 gives its home-decision diagnostics.

D.4.1 Three-model home-decision diagnostics

The comparison uses the same 512 source TEST request identities across models, retokenized for each model. Full prompts are retained up to 1,024 tokens on OLMoE and 512 on Qwen and DeepSeek. Every comparison uses batch size 64, a maximum of ten sequences per GPU, and $\gamma = 1.25$. EPLB and the complete hypergraph search use training requests; shared $\tau = 0.1$ is fixed before test timing. Three paired layout/run seeds (0–2) share predictor checkpoint seed 1234. Each condition uses two warmups and five timed passes; the median of those passes contributes one observation per seed. Configuration order rotates across seeds. Timing includes all eight batches and pipeline drain. The

runtime uses BF16, NVSHMEM 3.7.2, and nvshmem4py 0.3.1; all conditions share the same backend and model-specific communication calibration.

For the comparison of HG+PR with EPLB in Table 3, dispatch payloads decrease by 19.93 %, 23.17 %, and 28.76 % on OLMoE, Qwen, and DeepSeek, respectively. Reductions average the three paired per-seed percentages.

Sends count useful inter-node dispatch payloads, excluding padding. Complete prefill times for these policies are in Table 3; their differences cannot be attributed to send count alone. The offline actual-route control in Table 2 uses a separate fixed-layout request set and is not a deployable timing baseline for Table 3.

Native checkpoint numerical checks. For each model, seed 20270916 selects 128 test-set code requests with at least 288 tokens, supplying a 256-token prefix and 32 original continuation tokens. Each tokenizer determines eligibility; all execution variants use identical requests within that model. Teacher-forced continuations compare native execution with the original BF16 model. Paired 95% NLL intervals use 2,000 request resamples; every interval lies inside the predefined ± 0.01 nats/token tolerance. These checks validate numerical agreement of the tested static configurations; they do not measure downstream task quality.

Model	Layout	Homes	Δ NLL	Paired 95% CI
OLMoE	Load-only	Balanced	+0.00064	$[-0.00051, +0.00174]$
OLMoE	Hypergraph	Balanced	-0.00017	$[-0.00112, +0.00073]$
OLMoE	Hypergraph	Pre-Router	+0.00054	$[-0.00050, +0.00153]$
Qwen1.5-MoE	Load-only	Balanced	-0.00050	$[-0.00133, +0.00037]$
Qwen1.5-MoE	Hypergraph	Balanced	-0.00036	$[-0.00134, +0.00067]$
Qwen1.5-MoE	Hypergraph	Pre-Router	-0.00026	$[-0.00118, +0.00068]$
DeepSeek-V2-Lite	Load-only	Balanced	-0.00006	$[-0.00105, +0.00094]$
DeepSeek-V2-Lite	Hypergraph	Balanced	+0.00051	$[-0.00058, +0.00157]$
DeepSeek-V2-Lite	Hypergraph	Pre-Router	-0.00010	$[-0.00132, +0.00117]$

Table 12: Native three-model continuation checks against the original model. The hypergraph layout uses shared $\tau = 0.1$ and seed 0.

D.5 Migration and drift diagnostics

D.5.1 Migration mapping and transfers

Eight consecutive 64-request windows in a code–mathematics–code stream yield 21 successive old–target layout pairs across three seeds. Each window uses a complete load-anchor search with shared-τ candidates; $\tau = 0.1$ selects the target for this control. We compare fixed node labels, min-volume, and min-time mapping on every pair. Each policy then matches target GPU allocations to physical GPUs within each node to retain the most weights. The node and GPU mappings are shared across all layers. With Matching rounds, min-volume reduces remote bytes, endpoint peak bytes, and transfer time by 11.95 %, 12.01 %, and 10.47 % relative to Fixed. Min-time reduces them by 11.53 %, 12.85 %, and 11.82 %, respectively, and reduces time by another 1.49 % versus min-volume.

Each move transfers a complete 12 MiB BF16 OLMoE expert within a 256 MiB per-GPU receive budget. Serial moves one expert per round. Greedy scans moves in fixed order and admits up to three compatible remote moves; Matching uses edge coloring to attain the endpoint round bound (Appendix C.2). Remote rounds allow at most one send and one receive per node; subsequent local rounds apply the same rule to GPU endpoints. Each logical node has one 400 Gb/s leader port, without cross-port striping. At most four ports send concurrently, while Serial uses one. The effective payload rate is 40 decimal GB/s per port, below the 50 GB/s nominal rate. Timing starts before the first remote round and ends after the last local round; publication and inference overlap are excluded.

An untimed pass checks every transferred tensor. After two warmups, five timed repetitions run in rotated configuration order; we report their median full-transfer time. Enumeration checks the mapping objectives, and schedule replay checks move coverage, endpoint limits, and the round bound

for all 63 mapping cases. With Matching, min-time reduces transfer time by 1.49 % on average relative to min-volume, winning 16 of 21 pairs. Matching beats Greedy on all 21 pairs, saving 14.27–22.39%. With min-volume fixed, Matching cuts transfer time by 64.06 % versus Serial and 18.75 % versus Greedy. These comparisons hold the mapping fixed and use the same endpoint limits.

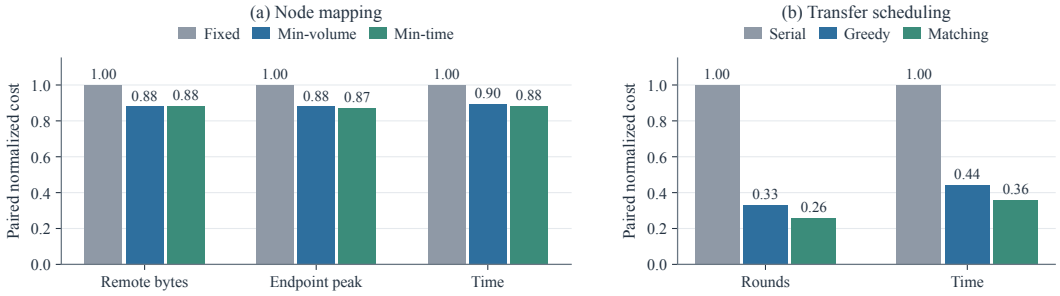

Figure 6: Mapping and scheduling on 21 layout pairs. Bars average paired ratios to Fixed (a) and Serial (b); lower is better. (a) fixes Matching rounds and within-node GPU matching; (b) fixes min-volume mapping.

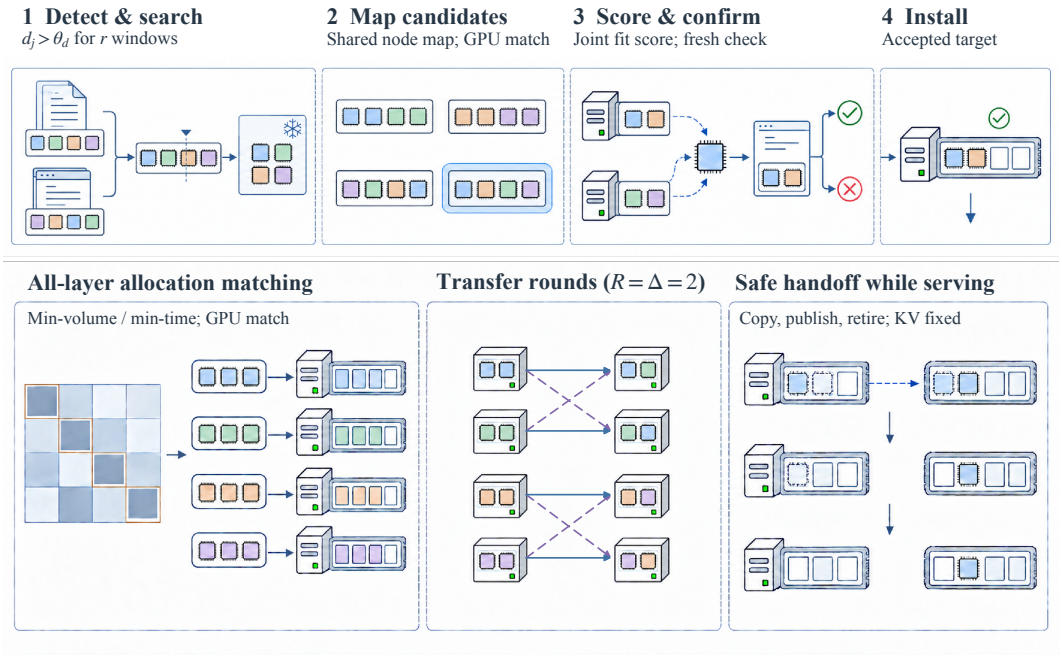

Figure 7: Online layout update. Top: detect drift, map candidates, score on W_{fit}, and confirm on fresh W_{check}. Bottom: match allocations, move experts in rounds with distinct endpoints, and publish safely while serving. Request homes and KV caches remain fixed.

D.5.2 Migration and service continuity

Round-wise IB migration controls. For OLMoE and three layout seeds, we migrate from the EPLB layout to the complete-search $\tau = 0.1$ target. Model-wide min-volume matching fixes the physical target before timing. Blocking, unrestricted online, and paced online policies share the same global expert-transfer plan, count-balanced homes, and 256 MiB/GPU extra-memory budget. Each round sends and receives at most one complete expert per node. Its expert-map updates and retirement finish before the next round starts, without waiting for all experts in a layer. All policies use the same expert storage and grouped computation path. The target-ready policy provides a reference without migration overhead.

Remote weight transfers and inference use separate NVSHMEM teams and workspaces over InfiniBand, with P2P disabled. Within each logical node, CUDA IPC copies forward weights over NVLink. The testbed remains one physical eight-H100 server with four logical nodes. A separate exact-copy probe checks weights and maps after every round while inference traffic continues; hardware NIC counters confirm that remote payloads traverse the fabric. The mapping controls in Figure 6 use the same four-port transfer configuration.

Each timed workload contains 512 test prompts truncated to at most 256 tokens, in 16 batches of 32. Two warmups precede five timed repetitions, with policy order rotated. The common clock starts before the first incoming expert is allocated. First-batch completion includes all 32 initial prefills; complete time includes all requests and any remaining transfers. The mean batch interval divides the time between first and last batch completion by the 15 intervening intervals, excluding startup and final migration drain. It measures foreground batch throughput while transfers proceed, rather than per-request TTFT. Tables report means of three seed medians; reductions average paired percentages. Exact target weights are checked after every timed pass, outside timing.

Model	Policy	Total	First	Migration	Drain	Extra	Done
OLMoE	Target ready	2050.7	128.7	0.0	0.1	—	—
OLMoE	Blocking	3004.4	1067.9	937.6	0.2	240	—
OLMoE	Online, unrestricted	2799.1	185.3	2540.7	0.2	240	15/15
OLMoE	Online, paced	2828.5	178.7	2769.9	114.8	240	—

Table 13: Round-wise IB migration. Times are in ms and Extra in MiB. Total includes final drain; First is first-batch completion. Drain is time after the last prefill, including final synchronization. Migration includes per-round allocation, transfers, map updates, and retirement. Extra includes resident growth, incoming copies, and transport workspaces; Done counts online migrations completed before final drain, out of 15 timed passes.

Transfer pacing. The paced online policy limits cumulative send and receive bytes to 10 GB/s per node and direction, allowing one expert-round burst. Unrestricted and blocking transfers have no software rate cap, but share the same 400 Gb/s (50 GB/s peak) InfiniBand ports with inference; this port rating is not an achieved migration rate. With the paced cap, the expected mean batch interval is 169.0 ms, versus 173.9 ms without software pacing. With migration and total time held at 2.770 and 2.829 s, respectively, earlier prefill completion leaves a 114.8 ms final drain. Thus pacing protects foreground service but does not improve total completion on these requests.

D.5.3 Long-horizon end-to-end adaptation

The 30-minute Code–Math–Code test in Figure 4 starts after 128 Code requests have established the initial reference. The stream then has mean arrival rate one request per second, with request-type switches at 4.8 and 18.7 minutes. Monitoring uses consecutive, non-overlapping 64-request windows and two threshold exceedances. The periodic controller checks every 512 seconds; the frozen controller never searches. Both updating policies perform complete shared-τ searches without a per-layer time limit. Candidate mapping and selection follow search; a separate 128-request window collected after the snapshot confirms the selected layout. The reference refreshes after confirmation. Accepted expert weights move over InfiniBand without software pacing, with node-local movement over NVLink; serving continues until the transfer-complete handoff. Panel (d) reports search, mapping–selection–confirmation, and complete weight-transfer time separately. Panel (c) accumulates the P95 TTFT differences in panel (b) at the stream's mean arrival rate.

The two request-type switches occur at 4.8 and 18.7 min. Two successive distances above the displayed 0.60 threshold start searches at 7.30 and 21.03 min. The searches last 208 and 222 s, respectively. Mapping, selection, and fresh-window confirmation add 2.13 and 2.38 s; uncapped InfiniBand weight transfer adds 2.54 and 2.59 s. The accepted layouts finish at 10.83 and 24.81 min. Requests retain the published layout until each safe handoff. Confirmation refreshes the reference before transfer completion, but panel (a) first shows the lower distance at a later completed window. Lines connect observed window endpoints; open squares mark first post-transfer samples. The double slash in panel (d) locally expands short post-search stages tenfold.

The triggered policy installs the Math layout before the first periodic update and reduces P95 TTFT earlier. After the workload returns to Code, both updating policies initially retain Math layouts, so cumulative savings fall until Code layouts are ready. The periodic policy rejects an intervening candidate and finishes transferring its next accepted layout near cutoff. The triggered policy recovers sooner and uses two searches versus three, ending with greater cumulative P95-based TTFT savings at 30 minutes. The plotted stream averages one request per second; panel (c) integrates the panel (b) P95 differences at that rate.

D.5.4 A ROUTE SHIFT INVISIBLE TO PAIR COUNTS

Consider eight experts in four pairs, $(0, 1), (2, 3), (4, 5), (6, 7)$, with four slots on each of two nodes. Every token chooses one expert from each pair. The eight selections with an even number of second members form the first phase; the eight odd-parity selections form the second. Every expert appears four times and every cross-pair expert pair appears twice in both phases. Thus single-expert loads and pairwise co-activation are identical, while the complete selected-set distributions have disjoint support.

Place the first member of each pair on Node 0 and the second on Node 1. Mean node visits rise from 1.75 to 2.00 after the shift. Swapping Experts 6 and 7 restores 1.75 visits; enumerating all 70 balanced labeled layouts verifies optimality in the respective phases. Both layouts have the same node load, and every fixed layout has unchanged pairwise cost across the shift. Averaging both request homes, useful dispatch copies over the eight selections change from 7 to 8 and back to 7 after the layout update, a 12.5% reduction. This is an exact constructed control, separate from the end-to-end stream in Figure 4.

D.5.5 NUMERICAL CONSISTENCY AND MODEL QUALITY

On 128 Code and 128 Math TEST requests, teacher-forced continuation uses a 256-token prefix and 32 original continuation tokens. For hypergraph placement with Pre-Router, mean NLL differences from the original BF16 model are -0.00030 and -0.00032 nats/token on Code and Math. The respective request-paired 95% intervals are $[-0.00141, 0.00083]$ and $[-0.00138, 0.00070]$, inside the predeclared ± 0.01 tolerance. The IB migration study separately verifies exact target weights, retained KV caches, and retirement of old copies.