Replicas-as-Variables: A Planner for Throughput-Optimal Routing in LLM Deployment

Shaojiang Wang[1,2] **Geer Yang**[3,1,4,5] **Wenbo Wang**[1]
Pengcheng Wang[6] **Bin Wu**[1†]

[1]Institute of AI for Industries, Chinese Academy of Sciences, Nanjing, China
[2]Zhongke Yuanma (Chengdu) Service Robot Research Institute Co., Ltd., Chengdu, China
[3]Institute of Computing Technology, Chinese Academy of Sciences, Beijing, China
[4]University of Chinese Academy of Sciences, Beijing, China
[5]Nanjing Institute of Software Technology, Nanjing, China
[6]Department of Computer Science and Technology, Tsinghua University, Beijing, China
seuhardware@aliyun.com, geer@iaii.ac.cn, wbwang@iaii.ac.cn
wpc20@tsinghua.org.cn, wubin@iaii.ac.cn

Abstract

Deploying large language models on heterogeneous GPU clusters requires coordinating memory placement, compute allocation, and communication. Fixed pipeline and replication templates restrict these decisions before the hardware bottlenecks are known. We introduce ReVar, a planner that treats component replica counts as decision variables for both dense stages and mixture-of-experts modules. A mixed-integer linear program jointly selects replicas, execution rates, and activation flows on a modular unit graph. The formulation provides throughput bounds under a profiled resource model, includes common deployment templates as restricted cases, and admits allocation laws in compute- and bandwidth-dominated regimes. Evaluation combines controlled topology simulation on an eight-H200 platform with experiments on six-node and nine-node heterogeneous clusters. On the nine-node, 38-GPU cluster, CUDA/NCCL execution for memory-fragmented Mixtral-8x22B improves throughput by approximately 22% over the strongest feasible comparator. Together, these results characterize when flexible replication and routing improve resource utilization and how the resulting plans connect to execution.

1 Introduction

Heterogeneous GPU fleets are a common substrate for LLM serving. For text generation and multimodal generative applications [26], inference throughput is constrained by GPU compute, VRAM, and interconnect bandwidth. Cloud availability and successive hardware generations create mixed fleets with fast local links, slower network boundaries, and uneven memory capacity [2; 6; 19; 23]. A deployment must decide how much work each device performs and which links carry intermediate activations.

Runtime systems improve batching, KV management, and request scheduling on a deployed model [27; 11; 30]. The preceding planning decision is equally consequential: given the model

†Corresponding author.

40th Conference on Neural Information Processing Systems (NeurIPS 2026).

graph and available GPUs, which components should be replicated, where should their copies reside, and how should traffic be split among them? A fixed pipeline commits to stage replicas and boundary links together. Full replication can exploit independent devices when the model fits, but cannot pool fragmented memory. Expert-parallel layouts expose replication choices for MoE experts [20; 13]; dense bottleneck components require the same flexibility.

ReVar expresses these choices in one placement-and-routing program. It represents a model as a unit graph of dense blocks, experts, and activation transfers, and optimizes component copies jointly with per-device execution rates and directed flows. The expanded space permits uneven dense replication and multiple links across stage boundaries while retaining conventional templates as restricted cases. A homogeneous fleet remains a valid choice whenever independent replicas or same-type pipelines use the available resources well.

We establish NP-hardness, derive fixed-placement throughput certificates, and characterize compute allocation and two-stage routing in compute- and bandwidth-dominated regimes. Sequence-affine execution keeps each request's dense path and KV state together; aggregate route shares govern admission across requests. The evaluation combines controlled topology simulation and two physical heterogeneous clusters, followed by planning-time scaling and auxiliary Transformer decoding. Our contributions are:

1. A unit-graph formulation that makes replication of dense and expert components part of joint placement and routing.
2. Throughput certificates and structural results that explain the role of resource budgets, template restrictions, and replica allocation.
3. Three complementary deployment studies, planning-time scaling, and analyses of graph granularity, demand adaptation, and stateful request control.

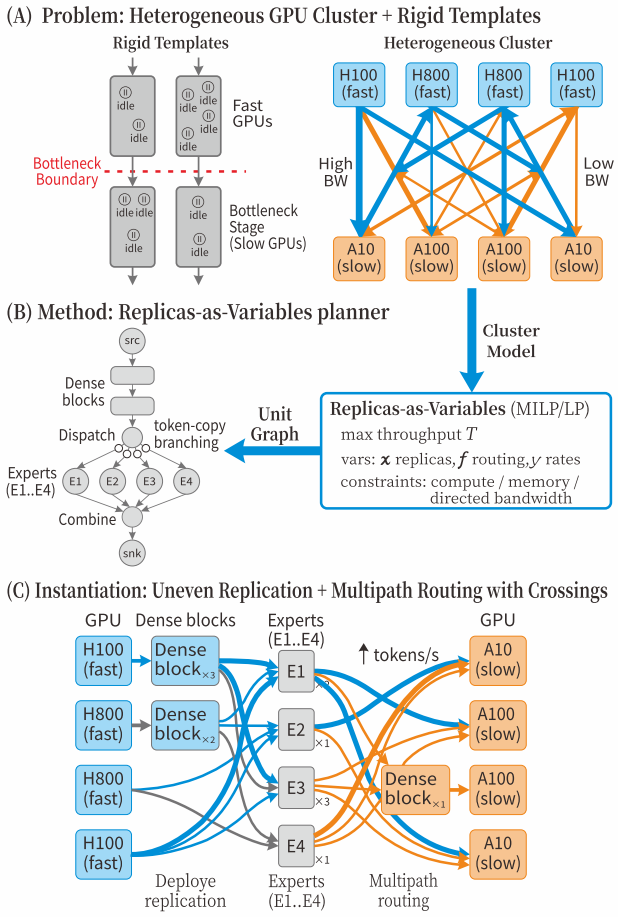

Figure 1: *ReVar at a glance.* A unit graph and profiled GPU/link budgets determine component placement, replica rates, and activation routes.

2 Related Work

Serving engines and request scheduling Serving systems such as Orca, vLLM, and DistServe improve goodput via batching, iteration-level scheduling, and KV-cache management for autoregressive decoding [27; 11; 30; 1; 24; 4]. These techniques typically assume a deployment template and are agnostic to heterogeneous module placement. ReVar is complementary: it addresses the upstream planning problem of placing replicated modules and routing activations under per-GPU compute and memory budgets and directed interconnect constraints.

Heterogeneous inference planning Pipeline-style templates remain core building blocks for scaling deep networks across devices [8; 18; 22]. Recent systems automate parallelization decisions in large-scale settings [29; 14] and, for heterogeneous resources, explicitly optimize deployments subject to multiple budgets [16; 10; 25; 15]. Helix jointly optimizes contiguous-layer placement and max-flow-based request routing, including intersecting per-request pipelines [16]; Parallax likewise composes request paths across layer replicas [25]. ReVar extends flow-based planning to modular dense/MoE unit graphs, with explicit component replica counts, per-unit resource accounting, and throughput certificates.

MoE inference and replication beyond experts MoE architectures [21; 12; 3] expose natural replication decisions at the expert granularity. Accordingly, prior work emphasizes expert parallelism and expert caching/replication to reduce communication and stragglers [20; 9; 7; 5; 13; 17]. In contrast, ReVar extends bottleneck-driven replication to arbitrary units, coupling dense stages, dispatch/gather, and experts in a single placement-and-routing program optimized end-to-end.

3 Methodology

ReVar models steady-state inference as resource allocation plus routing on a modular computation graph: each compute block (dense stage, MoE component, pre/post-processing, etc.) is a unit that can be replicated on any subset of GPUs, and edges represent activation transfer via explicit routing variables. This abstraction removes dependence on fixed deployment templates—pipeline parallelism, full replication, and expert parallelism (Appendix F, Proposition F.9). ReVar then maximizes throughput within the profiled compute, memory, and directed-bandwidth budgets.

3.1 Model and notation

Hardware budgets. Let G be the set of GPUs. Each GPU g has memory budget M_g (bytes) and compute budget P_g (FLOPs/s). Each directed link $g \to h$ has bandwidth budget $B_{g,h}$ (bytes/s); disallowed communication is represented by $B_{g,h} = 0$, and on-device transfer by $B_{g,g} = +\infty$. The budget M_g is the memory capacity available to deployment decisions after subtracting system reservations. To account for transient allocations and fragmentation, we reserve a factor $\rho \geq 0$: one weight copy of size W_u consumes $(1+\rho)W_u$ bytes, plus an activation/workspace term defined below.

Unit graph. The model is a directed acyclic graph (U, E) with a unique source `src` and sink `snk`; every unit lies on a source-to-sink path. Source and sink events are normalized to one model token. A *model token* (*tok*) is the unit of service-level throughput. A *token copy* (*tokc*) is the internal work unit that flows through the unit graph; one *tok* may induce multiple *tokc* objects in branched graphs (e.g., MoE). Service throughput is T in *tok*/s; all internal processing and routing rates are in *tokc*/s.

Each unit $u \in U$ is a deployable compute block with parameters: weight size W_u (bytes), per-*tokc* compute F_u (FLOPs/*tokc*), *input multiplicity* c_u^{in} (input *tokc* consumed per output *tokc* event at u), and activation/workspace footprint a_u (bytes per in-flight *tokc*). Each edge $u \to v \in E$ carries activations with *output multiplicity* $\alpha_{u\to v}^{\text{out}}$ (*tokc* emitted on edge $u \to v$ per output *tokc* event at u) and size $S_{u\to v}$ (bytes/*tokc*). With this convention, all conservation constraints are dimensionally closed: y and f are *tokc*/s rates, while c^{in} and α^{out} are *tokc*-to-*tokc* multiplicities (dimensionless). We allow c^{in} and α^{out} to be non-integers when they encode *expected* branching/merging under stochastic components (e.g., MoE gating); the program models steady-state rates in expectation.

Figure 2 illustrates the dense and MoE units and their token-copy flows.

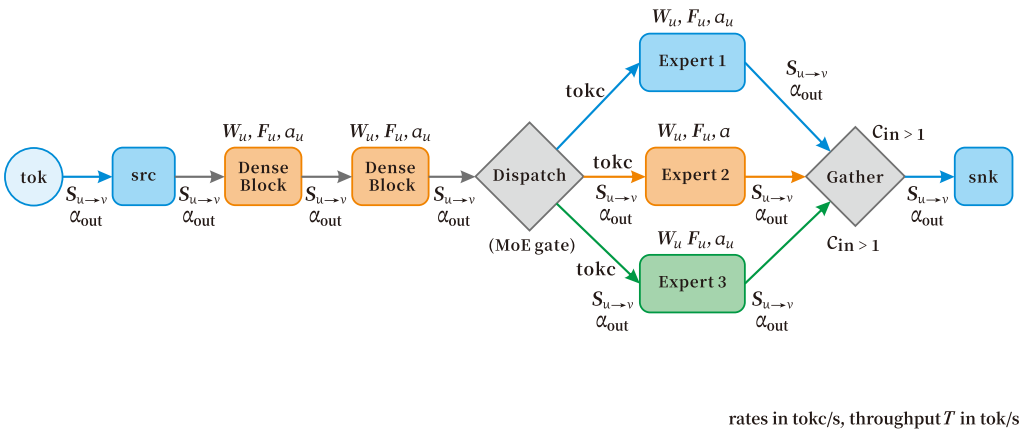

Figure 2: Token-copy semantics on a modular dense/MoE graph. Edge output multiplicities α^{out} and node input multiplicities c^{in} account for branching and merging. Throughput T is in *tok*/s; internal execution and routing rates are in *tokc*/s.

Multiplicity (global work per token). Multiplicity m_u is the number of *tokc* events induced at unit u by one model token in steady state. It is defined by the linear balance equations (for all $v \neq \texttt{src}$)

$$c_v^{\text{in}}\, m_v = \sum_{u \to v \in E} \alpha_{u\to v}^{\text{out}}\, m_u, \quad m_{\texttt{src}} = 1. \tag{1}$$

When (U, E) is a DAG and $c_v^{\text{in}} > 0$ for $v \neq \texttt{src}$, m is uniquely determined and can be computed by one topological pass (Appendix F, Lemma F.3). Multiplicity provides the *per-tok* work summary: per-token compute is $m_u F_u$ and per-token communication on $u \to v$ is $m_u \alpha_{u\to v}^{\text{out}} S_{u\to v}$. We model steady-state inference as token-copy flow on a modular unit graph, where branching is represented explicitly and multiplicities summarize global work per token.

3.2 Throughput upper bound program

Figure 3 visualizes how placement, execution rates, and routing flows jointly couple compute, memory, and directed-bandwidth constraints in a single program. We jointly choose unit replication, per-GPU *tokc* event rates, and flows between replicas. Units are constructed to fit a GPU, with tensor-parallel partitioning specified during graph construction. Binary per-device placement captures each GPU's shared compute budget (Appendix A). Decision variables are: $x_{u,g} \in \mathbb{Z}_{\geq 0}$ (copies of unit u on GPU g), $y_{u,g} \geq 0$ (*tokc*/s *output events* of u executed on g), $f_{u\to v,g,h} \geq 0$ (*tokc*/s routed from (u, g) to (v, h)), and throughput $T \geq 0$. Let $\tau_{u,g} > 0$ be the profiled service time (seconds) per *tokc* for running u on g; F_u/P_g is its FLOP-based specialization. Shape-specific measurements can be used directly. We maximize throughput subject to linear resource and flow constraints:

For multi-operand DAGs, pooled input balance specifies aggregate rates; execution matches branch identities and requests. Dense unit-multiplicity chains admit a request-path decomposition. Memory reservations account for queue storage, retained activations, and resident KV alongside in-service occupancy. For fixed placement x, optimizing over (y, f, T) is a linear program (Appendix F, Proposition F.7); in general the MILP is NP-hard and has $\Theta(|U||G| + |E||G|^2)$ variables (Appendix F, Theorem F.12).

Appendix A provides latency-SLO and KV-memory extensions, scalable decomposition, and solver settings.

The activation/workspace term $a_u \tau_{u,g} y_{u,g}$ represents average in-service occupancy by Little's law (Appendix F, Lemma F.4). The feasible set describes steady-state *averages*: `src` injects work at rate T (hence it has no node-input constraint), and all other units consume/produce *tokc* at the rates dictated by c^{in} and α^{out}. The optimum is an upper bound for executions whose average resource demands satisfy this model (Appendix F, Theorem F.5). With interchangeable fluid inputs and separately provisioned buffers, feasible rates admit a causal periodic schedule (Proposition F.6).

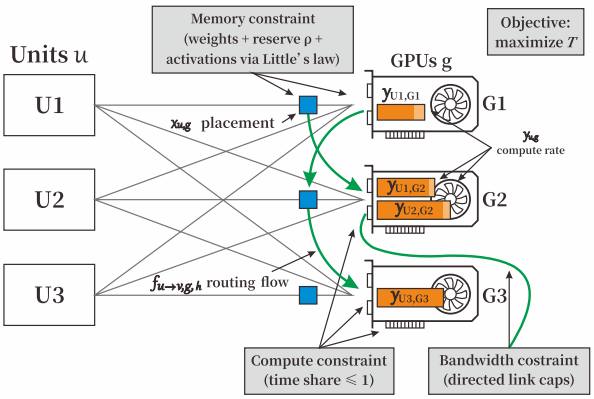

Figure 3: Coupled throughput program: Placement $x_{u,g}$ activates execution rates $y_{u,g}$, while routing $f_{u \to v,g,h}$ distributes token flows within bandwidth limits.

3.3 ReVar as the Solver

ReVar. ReVar is the deployment procedure induced by the unit-graph model and program (2): build (U, E) and profile (W, F, a, S), solve the MILP, then extract a placement and aggregate rate plan.

$$
\begin{aligned}
\max_{x,y,f,T} \quad & T \\
\text{s.t.} \quad & \sum_u (1+\rho) W_u x_{u,g} + \sum_u a_u \tau_{u,g} y_{u,g} \le M_g && \forall g, \\
& \sum_u \tau_{u,g} y_{u,g} \le 1 && \forall g, \\
& y_{u,g} \le x_{u,g} / \tau_{u,g} && \forall u, g, \\
& \sum_h f_{u \to v,g,h} = \alpha^{\text{out}}_{u \to v} y_{u,g} && \forall (u \to v) \in E,\ g, \\
& \sum_{u \to v \in E} \sum_g f_{u \to v,g,h} = c^{\text{in}}_v y_{v,h} && \forall v \ne \texttt{src},\ h, \\
& \sum_{u \to v \in E} S_{u \to v} f_{u \to v,g,h} \le B_{g,h} && \forall g, h, \\
& \sum_g y_{\texttt{src},g} = \sum_g y_{\texttt{snk},g} = T, \\
& x_{u,g} \in \mathbb{Z}_{\ge 0}, \quad y_{u,g}, f_{u \to v,g,h}, T \ge 0.
\end{aligned}
\tag{2}
$$

The inequalities account for memory, shared GPU service time, replica activation, and directed-link capacity; the equalities preserve edge output, node input, and model-token throughput.

By constraining (x, f), the same formulation contains restricted pipeline, full-model replication, and expert-parallel templates (Appendix F, Proposition F.9).

Restricted templates and path realization. A Helix-style chain restriction uses binary one-stage-per-GPU placement; it is contained in the ReVar feasible set under the same coefficients (Appendix F, Theorem F.11). For unit-multiplicity chains, residual-flow decomposition extracts path weights from the optimized flows (Appendix F.4).

Deployment-aware selection. On profiled hardware, a shared runtime calibrator ranks feasible candidates from the MILP and restricted templates using compute-fragmentation and communica-

tion features. Selection maximizes predicted throughput over these candidates; Section 5.3 evaluates this procedure with leave-one-model-out calibration.

Stateful execution. For autoregressive decoding, admission selects a dense path and pins the sequence and its KV state to that path. Weighted-deficit admission tracks the planned aggregate shares across requests; per-path batching, queue credits, and backpressure limit outstanding work. Changes in load can update admission weights after solving the fixed-placement LP while existing sequences retain their paths. Appendix A gives linear KV and shared-fabric extensions.

4 Theory and Special Cases with Closed-form Solutions

The resource program bounds throughput under its stated steady-state accounting assumptions. For interchangeable fluid inputs, its rates admit a periodic time-sharing schedule with separately provisioned buffers; request-level joins and state affinity are enforced by the runtime (Appendix F). The placement problem is NP-hard, even for a dense chain with two GPUs. The following special cases isolate how compute and link capacities shape replica allocation.

4.1 Per-token summaries

Let $Y_u = \sum_g y_{u,g}$. On an acyclic unit graph with well-defined multiplicities, flow balance gives

$$Y_u = m_u T, \qquad \forall u \in U. \tag{3}$$

The resulting compute and communication demands per model token are

$$A_u = m_u F_u, \qquad D_{u\to v} = S_{u\to v}\alpha^{\text{out}}_{u\to v} m_u. \tag{4}$$

Write $G[u] = \{g : x_{u,g} \geq 1\}$ and $r_u = |G[u]|$. For these closed-form regimes, service times specialize to $\tau_{u,g} = F_u/P_g$, and memory is weight-only ($a_u = 0$, $\rho = 0$), with each stated placement fitting its weight budget. Additional memory reservations can be retained whenever they remain nonbinding for the constructed solution.

4.2 Compute-dominated laws

Full replication. If the whole model fits on every GPU, $\sum_u W_u \leq M_g$, full replication with on-device routing attains

$$T^\star = \frac{\sum_g P_g}{\sum_u A_u}. \tag{5}$$

This saturates aggregate compute and needs no inter-GPU bandwidth (Theorem F.17).

Stage and expert replication. For N homogeneous GPUs of compute P, nonbinding bandwidth, and one stage per GPU, the optimal integer allocation satisfies

$$T^\star = \max_{\substack{\sum_u r_u = N \\ r_u \geq 1 \text{ integer}}} \min_u \frac{r_u P}{A_u} \;\leq\; \frac{NP}{\sum_u A_u}. \tag{6}$$

Equality holds when $r_u = NA_u/\sum_w A_w$ is integral for every stage. Thus the proportional law is exact when compatible with integer placement (Theorem F.18). For MoE experts, let p_e be the normalized expected selection frequency, so $m_e = kp_e$ for top-k gating. With R dedicated expert slots and sufficient non-expert capacity, the same law applies with $A_e = kp_eF_e$ and N replaced by R (Theorem F.19).

4.3 Bandwidth-dominated laws

Two-stage split across a slow boundary. Consider units $a \to b$ that each fit on a GPU but cannot co-locate. Allocate r of N homogeneous GPUs to a and the rest to b, with compute P and capacity B on each directed inter-GPU link. For boundary demand $D = D_{a\to b}$,

$$T[r] = \min\left\{\frac{rP}{A_a}, \frac{(N-r)P}{A_b}, \frac{r(N-r)B}{D}\right\}, \tag{7}$$

$$T^\star = \max_{1\leq r\leq N-1} T[r].$$

Uniform routing across the complete bipartite boundary attains $T[r]$ (Theorem F.20). When compute is nonbinding, a balanced split uses $\Theta(N^2)$ links and attains $\Theta(N^2B/D)$, compared with $O(NB/D)$ for disjoint fixed pairs (Corollary F.21; Figure 4). These laws concern independent pairwise link budgets; shared-fabric constraints are added as in Appendix A. In MoE graphs, dispatch and gather commodities share these budgets and are optimized jointly by the routing program.

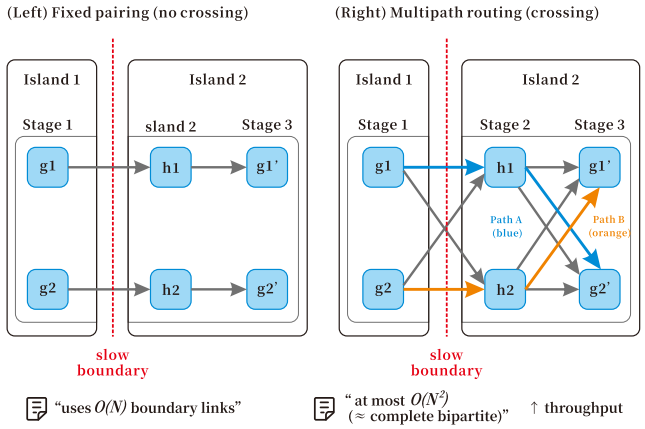

Figure 4: Fixed pairing versus multipath routing. Independent pairwise link budgets enable the illustrated gain; each sequence uses one path.

5 Experiments

We organize the deployment evaluation into three groups: (I) controlled topology simulation on a homogeneous eight-H200 platform; (II) a physical six-node cluster with L20 and P4 GPUs; and (III) a physical nine-node cluster spanning five GPU types. We then examine planning time in a repeated-pod scaling simulation. Within each group, methods share the same model graphs and resource budgets. System-named baselines implement their deployment templates in the common evaluation harness (Appendix B.1.3). Reported throughput counts normalized tokens.

5.1 Group I: simulation on a homogeneous GPU platform

An eight-H200 NVSwitch node provides a common physical platform for simulating six resource configurations through compute delays, VRAM reservations, and bandwidth controls. C1/C2 represent A100/H100 NVSwitch pods, C3 two A100 islands, C4 a mixed H100/A10 fleet, and C5/C6 memory-constrained three-device systems. Table 1 compares the resulting deployments. ReVar achieves the highest or tied-highest throughput in 14 of 15 settings. Its largest gain over the strongest comparator is 15.8% on Mixtral-8x7B/C4, where component replication and multiple routes distribute work across uneven resources. Appendices B and C.2 give the topology definitions, ablations, and utilization analysis.

5.2 Group II: six-node L20/P4 cluster

This physical cluster contains eight L20 GPUs and one P4 GPU across six nodes. Hardware profiling measures GPU memory, shape-specific GEMM rates, and network performance. The stage executor runs Gloo point-to-point communication on CPU tensors and uses GEMM-calibrated compute delays. Pipeline throughput is determined by the slowest stage, while latency sums the stage times. Table 2 covers all four evaluated models. ReVar achieves the highest throughput on DeepSeek-V2-Lite, Mistral-7B, and Mixtral-8x7B, including gains of 33.3% and 49.6% over the strongest comparators for the two MoE models. Appendix D.1 provides the measurement protocol.

5.3 Group III: nine-node cluster with five GPU types

The larger physical cluster spans four availability zones and contains 12 A10, eight V100, eight P100, eight T4, and two P4 GPUs. Planning uses shape-specific median GEMM rates, maximum contiguous memory allocations, P10 communication profiles, and shared node-pair/NIC capacity constraints. We evaluate 5% and 10% capacity headroom. The executor reserves the planned weight

Table 1: Group I: controlled topology simulation on the homogeneous H200 platform. Throughput is in normalized tokens/s; C1–C6 are defined in Table 4. **Bold** indicates the best result and underline indicates the second-best. "–" denotes infeasible / not supported.

Model	Cluster	ReVar	Greedy	Random	LLM-PQ	HG2	Helix
Llama-3-8B	C1	**58,192.7**	24,461.2	24,613.4	24,580.5	49,160.9	58,083.1
	C2	**184,463.3**	77,215.8	77,807.4	77,916.9	155,833.8	182,352.5
	C3	**58,192.7**	24,137.8	24,581.4	24,357.9	49,160.9	57,081.9
	C4	**103,888.8**	77,415.7	30,128.9	77,916.9	7,764.8	102,727.7
	C5	5,492.3	**5,502.3**	5,492.3	–	–	–
Mistral-7B	C1	**60,996.1**	29,103.1	29,099.9	29,152.7	58,199.8	60,791.0
	C2	**193,349.7**	92,252.1	92,341.9	92,646.8	184,485.8	192,537.8
	C3	**60,996.1**	29,208.1	29,079.3	29,034.4	58,149.5	60,488.6
	C4	**108,893.6**	92,313.5	32,053.8	92,141.7	103,901.5	105,581.4
	C5	**5,960.5**	4,051.3	5,951.4	–	–	5,851.1
	C6	**5,291.5**	5,242.1	**5,291.5**	–	–	–
Mixtral-8x7B	C1	**36,115.5**	30,994.4	25,301.6	17,558.2	35,116.5	–
	C2	**114,490.1**	98,248.3	80,202.7	55,661.5	111,323.0	–
	C3	**36,115.5**	30,994.4	25,301.6	17,558.2	35,116.5	–
	C4	**64,480.3**	55,523.4	12,417.6	55,452.3	55,661.5	–

Table 2: Group II: six-node L20/P4 results. Rate: normalized tok/s; latency: summed stage time (ms). Bold/underline mark the highest/second-highest rates. ReVar and Helix share the Llama-3-8B MILP optimum of 43,457.65 tok/s; their throughput difference reflects execution-level timing effects.

	DeepSeek-V2-Lite		Llama-3-8B		Mistral-7B		Mixtral-8×7B	
Method	Rate	Latency	Rate	Latency	Rate	Latency	Rate	Latency
ReVar	**29,550**	30.70	35,190	25.05	**37,851**	20.47	**16,432**	52.70
Helix	–	–	**36,969**	18.78	37,350	23.90	–	–
HexGen-2	22,160	37.07	19,325	36.17	18,116	33.42	10,163	88.38
Pipeline Colocation	13,494	62.73	13,126	56.65	14,814	52.37	8,388	119.43
Trunk Replication	18,037	37.22	–	–	–	–	8,940	73.13
Greedy	8,139	77.65	13,181	56.86	13,023	55.01	10,982	64.32

memory, runs shape-equivalent cuBLAS operations, and transfers GPU tensors over the planned NCCL routes.

The deployment-aware procedure scores feasible ReVar and restricted-template candidates with a shared calibrator using compute fragmentation, message sizes, and topology features. Leave-one-model-out calibration excludes all runtime labels of the target model. Table 3 reports the measured throughput of the selected deployments, which attain the highest candidate throughput in all eight model–headroom groups. For Mixtral-8x22B, whose BF16 weights exceed the effective memory of each individual GPU-type pool, flexible replication and routing combine resources across pools and improve throughput by approximately 22% over the strongest feasible comparator at both headrooms. Appendix D.2 details the profiling, execution, and calibration procedure.

5.4 Planning time and scalability

Figure 5 plots solver wall time against the number of decision variables for dense and MoE instances, including both MILP solves and LP relaxations. The measurements characterize how model size and solver formulation affect planning cost.

We also simulate fleets of 8–100,000 GPUs as repeated, independent eight-GPU C2-style pods. Summing measured per-pod MILP times gives serial planning-time estimates of 0.183 s for Llama-3-8B

Table 3: Group III: nine-node throughput (normalized tok/s) at 5%/10% headroom. Bold/underline mark the highest/second-highest rates, including ties. ReVar is defined in Section 5.3. N/A: inapplicable; –: no feasible deployment.

	Llama-3-8B		Mistral-7B		Mixtral-8×7B		Mixtral-8×22B	
Method	5%	10%	5%	10%	5%	10%	5%	10%
ReVar	**58,702**	**58,702**	**59,959**	**51,505**	**18,555**	**18,555**	**4,479**	**4,480**
Full Replication	**58,702**	**58,702**	**59,959**	37,484	N/A	N/A	N/A	N/A
Helix	36,138	39,396	17,135	17,289	N/A	N/A	N/A	N/A
HexGen-2	37,901	37,901	51,505	**51,505**	**18,555**	**18,555**	–	–
LLM-PQ	13,509	13,509	15,695	15,695	9,703	9,703	–	–
Pipeline Colocation	13,486	13,486	16,205	16,205	10,015	10,015	–	–
Trunk Replication	N/A	N/A	N/A	N/A	8,910	10,070	–	–
Greedy	13,923	13,923	15,699	15,699	5,023	5,023	1,656	1,673
Random Best-of-3	14,475	14,475	17,074	17,074	2,914	2,914	1,164	1,164
MILP-NoRep	13,509	13,509	15,695	15,695	9,703	9,703	3,671	3,671
MILP-NoExpRep	N/A	N/A	N/A	N/A	9,703	9,703	3,330	3,328

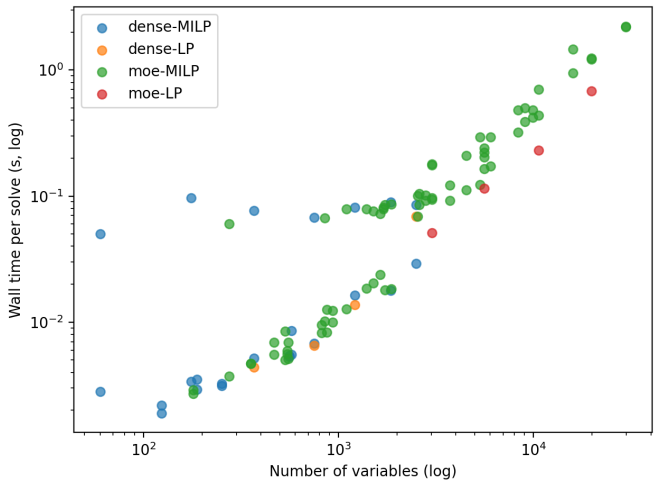

Figure 5: Solver wall time versus decision-variable count on logarithmic axes. Each point is a solver call; colors distinguish dense/MoE instances and MILP/LP formulations.

and 54.56 s for Mixtral-8x7B at 1,024 GPUs. Planning time and constraint count scale linearly with the number of independent pods (Appendix B.5).

Graph granularity provides an additional control on planning cost. Across C1–C4, a four-group Mixtral graph retains at least 99.9924% of the one-group-per-layer objective, with maximum solver time 2.227 s versus 39.795 s for the layer-level graph (Appendix E.1). Further analyses cover sparse route construction and demand adaptation. Auxiliary Transformer decoding and resident-KV control experiments are collected in Appendices E.4 and E.5.

6 Conclusion

ReVar jointly optimizes component replication, placement, and activation routing for heterogeneous LLM deployment. Its unit-graph resource model connects deployment choices to throughput certificates and interpretable allocation laws. Controlled topology simulation and two physical-cluster studies show how flexible replication and routing exploit heterogeneous resources. Planning-time

simulations characterize decomposed scaling, while graph-granularity and stateful-decoding studies connect efficient planning to practical execution.

Acknowledgments and Disclosure of Funding

This work was supported by the Frontier Technologies R&D Program of Jiangsu (Grant No. BF2024052), the Department of Science and Technology of Jiangsu Province (Grant No. BM20250016 and Grant No. BM20250019), Nanjing Municipal Science and Technology Bureau (Grant No. 202512136), and Chengdu Science and Technology Program (Grant No. 2025-YF08-00097-GX).

The authors declare no other competing financial interests.

References

[1] Amey Agrawal, Nitin Kedia, Ashish Panwar, Jayashree Mohan, Nipun Kwatra, Bhargav S. Gulavani, Alexey Tumanov, and Ramachandran Ramjee. Taming throughput–latency tradeoff in LLM inference with Sarathi-Serve. *CoRR*, abs/2403.02310, 2024.

[2] Amazon Web Services. Amazon EC2 accelerated computing instance types. https://aws.amazon.com/ec2/instance-types/accelerated-computing/, 2026. Accessed: 2026-05-05.

[3] 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.

[4] Yao Fu, Leyang Xue, Yeqi Huang, Andrei-Octavian Brabete, Dmitrii Ustiugov, Yuvraj Patel, and Luo Mai. ServerlessLLM: Low-latency serverless inference for large language models. In *18th USENIX Symposium on Operating Systems Design and Implementation (OSDI 24)*, pages 135–153, Santa Clara, CA, 2024. USENIX Association.

[5] Trevor Gale, Deepak Narayanan, Cliff Young, and Matei Zaharia. Megablocks: Efficient sparse training with mixture-of-experts. In *Proceedings of Machine Learning and Systems*, volume 5, 2023.

[6] Google Cloud. GPU machine types. https://docs.cloud.google.com/compute/docs/gpus, 2026. Accessed: 2026-05-05.

[7] Jiaao He, Jiezhong Qiu, Aohan Zeng, Zhilin Yang, Jidong Zhai, and Jie Tang. Fastmoe: A fast mixture-of-expert training system. *CoRR*, abs/2103.13262, 2021. URL https://arxiv.org/abs/2103.13262.

[8] Yanping Huang, Youlong Cheng, Ankur Bapna, Orhan Firat, Mia X. Chen, Dehao Chen, HyoukJoong Lee, Jiquan Ngiam, Quoc V. Le, Yonghui Wu, and Zhifeng Chen. GPipe: Efficient training of giant neural networks using pipeline parallelism. In *Advances in Neural Information Processing Systems 32 (NeurIPS 2019)*, pages 103–112, 2019.

[9] Changho Hwang, Wei Cui, Yifan Xiong, Ziyue Yang, Ze Liu, Han Hu, Zilong Wang, Rafael Salas, Jithin Jose, Prabhat Ram, Joe Chau, Peng Cheng, Fan Yang, Mao Yang, and Yongqiang Xiong. Tutel: Adaptive mixture-of-experts at scale. *CoRR*, abs/2206.03382, 2022. URL https://arxiv.org/abs/2206.03382.

[10] Youhe Jiang, Ran Yan, and Binhang Yuan. Hexgen-2: Disaggregated generative inference of llms in heterogeneous environment, 2025. URL https://arxiv.org/abs/2502.07903. ICLR 2025.

[11] Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with PagedAttention. In *Proceedings of the 29th ACM SIGOPS Symposium on Operating Systems Principles (SOSP '23)*, pages 611–626, New York, NY, USA, 2023. ACM. doi: 10.1145/3600006.3613165.

[12] D. Lepikhin, H. Lee, Y. Xu, D. Chen, O. Firat, Y. Huang, M. Krikun, N. Shazeer, and Z. Chen. GShard: Scaling giant models with conditional computation and automatic sharding. In *International Conference on Learning Representations (ICLR)*, 2021. URL https://openreview.net/forum?id=qrwe7XHTmYb.

[13] K. Li, W. Huang, Q. Wang, L. Zheng, X. Liao, H. Jin, and J. Xue. Diff-moe: Efficient batched MoE inference with priority-driven differential expert caching. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC '25)*, pages 1951–1965. ACM, 2025. doi: 10.1145/3712285.3759903.

[14] Zhuohan Li, Lianmin Zheng, Yinmin Zhong, Vincent Liu, Ying Sheng, Xin Jin, Yanping Huang, Zhifeng Chen, Hao Zhang, Joseph E. Gonzalez, and Ion Stoica. AlpaServe: Statistical multiplexing with model parallelism for deep learning serving. In *17th USENIX Symposium on Operating Systems Design and Implementation (OSDI 23)*, pages 663–679, Boston, MA, July 2023. USENIX Association. ISBN 978-1-939133-34-2. URL https://www.usenix.org/conference/osdi23/presentation/li-zhouhan.

[15] Tao Luo, Kelvin K. W. Ng, Zhen Ping Khor, Sidharth Sankhe, Boon Thau Loo, and Vincent Liu. Multiplexed heterogeneous LLM serving via stage-aligned parallelism. In *ACM Symposium on Cloud Computing (SoCC '25)*, Online, USA, November 2025. ACM. doi: 10.1145/3772052.3772207. URL https://doi.org/10.1145/3772052.3772207.

[16] Yixuan Mei, Yonghao Zhuang, Xupeng Miao, Juncheng Yang, Zhihao Jia, and Rashmi Vinayak. Helix: Serving large language models over heterogeneous GPUs and network via max-flow. In *Proceedings of the 30th ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS '25)*, Rotterdam, Netherlands, 2025. ACM.

[17] V. P. K. Miriyala, G. Sviridov, B. Chen, and H. Javaid. Latency-optimal load balancing for distributed MoE inference. In *Proceedings of the 1st Workshop on Inter-Networking Challenges for AI (INET4AI '25)*, pages 7–13. ACM, 2025. doi: 10.1145/3769695.3771675.

[18] Deepak Narayanan, Aaron Harlap, Amar Phanishayee, Vijay Seshadri, Nikhil Devanur, Gregory R. Ganger, Phillip B. Gibbons, and Matei Zaharia. Pipedream: Generalized pipeline parallelism for DNN training. In *Proceedings of the 27th ACM Symposium on Operating Systems Principles (SOSP '19)*, pages 1–15, 2019. URL https://dl.acm.org/doi/10.1145/3341301.3359646.

[19] NVIDIA. Introduction to NVIDIA DGX H100/H200 systems. https://docs.nvidia.com/dgx/dgxh100-user-guide/introduction-to-dgxh100.html, 2026. Accessed: 2026-05-05.

[20] 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 (ICML)*, volume 162 of *Proceedings of Machine Learning Research*, pages 18332–18346, 2022. URL https://proceedings.mlr.press/v162/rajbhandari22a.html.

[21] 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 *Proceedings of the 5th International Conference on Learning Representations (ICLR)*, 2017. Poster.

[22] Mohammad Shoeybi, Mostofa Patwary, Raul Puri, Patrick LeGresley, Jared Casper, and Bryan Catanzaro. Megatron-lm: Training multi-billion parameter language models using model parallelism. In *arXiv preprint arXiv:1909.08053*, 2019.

[23] Foteini Strati, Paul Elvinger, Tolga Kerimoglu, and Ana Klimovic. ML training with cloud GPU shortages: Is cross-region the answer? In *Proceedings of the 4th Workshop on Machine Learning and Systems (EuroMLSys '24)*, pages 107–116. ACM, 2024. doi: 10.1145/3642970.3655843.

[24] Biao Sun, Ziming Huang, Hanyu Zhao, Wencong Xiao, Xinyi Zhang, Yong Li, and Wei Lin. Llumnix: Dynamic scheduling for large language model serving. In *18th USENIX Symposium on Operating Systems Design and Implementation (OSDI 24)*, pages 173–191, Santa Clara, CA, July 2024. USENIX Association. ISBN 978-1-939133-40-3. URL https://www.usenix.org/conference/osdi24/presentation/sun-biao.

[25] Chris Tong, Youhe Jiang, Gufeng Chen, Tianyi Zhao, Sibian Lu, Wenjie Qu, Eric Yang, Lynn Ai, and Binhang Yuan. Parallax: Efficient LLM inference service over decentralized environment. *CoRR*, abs/2509.26182, 2025.

[26] Pengcheng Wang, Haolun Peng, Siyang Mao, Shaojiang Wang, and Jie Tang. A review of generative models for virtual human motion driving. *Science China Information Sciences*, 68(8):181102, May 2025. ISSN 1869-1919. doi: 10.1007/s11432-023-4284-x. URL https://doi.org/10.1007/s11432-023-4284-x.

[27] Gyeong-In Yu, Joo Seong Jeong, Geon-Woo Kim, Soojeong Kim, and Byung-Gon Chun. Orca: A distributed serving system for transformer-based generative models. In *Proceedings of the 16th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, 2022.

[28] Juntao Zhao, Borui Wan, Yanghua Peng, Haibin Lin, and Chuan Wu. Llm-pq: Serving llm on heterogeneous clusters with phase-aware partition and adaptive quantization, 2024. URL https://arxiv.org/abs/2403.01136.

[29] Lianmin Zheng, Zhuohan Li, Hao Zhang, Yonghao Zhuang, Zhifeng Chen, Yanping Huang, Yida Wang, Yuanzhong Xu, Danyang Zhuo, Eric P. Xing, Joseph E. Gonzalez, and Ion Stoica. Alpa: Automating inter- and intra-operator parallelism for distributed deep learning. In *Proceedings of the 16th USENIX Symposium on Operating Systems Design and Implementation (OSDI '22)*, pages 559–578, 2022.

[30] Yinmin Zhong, Shengyu Liu, Junda Chen, Jianbo Hu, Yibo Zhu, Xuanzhe Liu, Xin Jin, and Hao Zhang. Distserve: Disaggregating prefill and decoding for goodput-optimized large language model serving. *CoRR*, abs/2401.09670, 2024.

A Extensions, certificates, and modeling details

This appendix develops fixed-placement certificates and linear extensions of the resource model. Throughout, $\tau_{u,g} > 0$, $a_u \geq 0$, and $S_{u\to v} > 0$ for modeled activation edges. We refer to the labeled constraints in Appendix F.1.

A.1 Scope and limitations

ReVar plans steady-state resource allocation from a modular graph and profiled service coefficients. Absolute serving performance also depends on batching, kernel choice, bursty arrivals, queueing, and synchronization. Coefficient profiles are therefore tied to a workload and runtime configuration. The aggregate flow equations describe expected work rates; an executor additionally preserves operand identity at joins and sequence affinity for stateful decoding.

The in-service activation term is an average occupancy model. Deployment memory also covers queue buffers, allocator workspaces, and resident KV. Shared NICs and fabrics can constrain several GPU pairs jointly, as captured by the contention-domain extension below. The integer program grows as $\Theta(|U||G| + |E||G|^2)$; restricted routing and LP relaxations reduce planning cost, with certificates stated for the precise feasible space that is solved.

Broader impacts. More efficient use of heterogeneous accelerators can lower deployment costs and extend the useful life of available hardware, broadening access to inference infrastructure. The same capacity gains can increase aggregate resource demand or reduce the cost of harmful applications of existing models. Energy outcomes depend on utilization, device efficiency, and workload growth; deployment planning complements application-level access controls and monitoring.

A.2 Certificates for fixed placement

For fixed placement, dual variables provide both throughput bounds and marginal values of resource budgets.

Theorem A.1 (LP dual and certificate for fixed placement)**.** *Fix an integer placement x satisfying $\sum_u (1+\rho) W_u x_{u,g} \leq M_g$ for every g. Set*

$$\widetilde{M}_g = M_g - \sum_u (1+\rho) W_u x_{u,g}, \qquad \widetilde{Y}_{u,g} = x_{u,g}/\tau_{u,g}.$$

Let $\mathcal{L} = \{(g,h) : B_{g,h} < \infty\}$; constraints and dual terms for infinite-capacity pairs are omitted. For nonnegative dual variables $\pi_g, \sigma_g, \gamma_{u,g}, \lambda_{g,h}$ and free variables $\phi_{u\to v,g}, \psi_{v,h}, \eta_{\mathrm{src}}, \eta_{\mathrm{snk}}$, the dual is

$$\min \sum_g \pi_g \widetilde{M}_g + \sum_g \sigma_g + \sum_{u,g} \gamma_{u,g} \widetilde{Y}_{u,g} + \sum_{(g,h)\in\mathcal{L}} \lambda_{g,h} B_{g,h}$$

subject to

$$\begin{aligned}
&a_u \tau_{u,g} \pi_g + \tau_{u,g} \sigma_g + \gamma_{u,g} - \sum_{v:u\to v} \alpha^{\mathrm{out}}_{u\to v} \phi_{u\to v,g} - \mathbb{I}[u \neq \mathrm{src}] c^{\mathrm{in}}_u \psi_{u,g} \\
&\qquad + \mathbb{I}[u = \mathrm{src}] \eta_{\mathrm{src}} + \mathbb{I}[u = \mathrm{snk}] \eta_{\mathrm{snk}} \geq 0, && \forall u, g, \\
&\phi_{u\to v,g} + \psi_{v,h} + S_{u\to v} \lambda_{g,h} \geq 0, && \forall (u \to v), g, h, \\
&-\eta_{\mathrm{src}} - \eta_{\mathrm{snk}} \geq 1.
\end{aligned}$$

Here $\lambda_{g,h} = 0$ for $(g,h) \notin \mathcal{L}$, and ψ is defined only for nonsource units. The fixed-placement primal and dual have equal optimal values. Every dual-feasible point bounds the primal optimum above by its displayed dual objective.

Proof. Use nonnegative multipliers for the memory, compute, activation-cap, and finite-bandwidth inequalities. Write each edge equality as $\sum_h f - \alpha y = 0$, each node equality as $\sum f - c y = 0$, and each boundary equality as $\sum_g y - T = 0$, assigning the free multipliers above. For the primal form $\max\{c^\top z : Az \leq b, Ez = 0, z \geq 0\}$, collecting coefficients gives $A^\top \xi + E^\top \zeta \geq c$ and the displayed dual. The zero-rate point is feasible because $\widetilde{M}_g \geq 0$; positive service times and finite GPU count bound T. LP strong duality and weak duality give the conclusions. □

Corollary A.2 (Concavity and budget sensitivity)**.** *For fixed x and fixed coefficient matrix, the finite optimal value is concave and piecewise linear in the right-hand-side budget vector. Any dual optimum is a supergradient: if $\xi^\star$ is optimal at b, then*

$$T^\star(b') \le T^\star(b) + (\xi^\star)^\top (b' - b)$$

for every other feasible budget vector b'.

Proof. The value is the pointwise minimum of the dual linear objectives $b^\top \xi$. Evaluating an optimizer at b on the new budget b' gives the inequality. The finite-dimensional LP value is piecewise linear on its feasible domain. □

A.3 Placement and routing structure

Proposition A.3 (Binary placement without loss of optimality)**.** *Within the time-sharing resource model, every feasible solution has an equal-throughput counterpart with $x_{u,g} \in \{0, 1\}$.*

Proof. Replace $x_{u,g}$ by $\mathbb{I}[x_{u,g} \ge 1]$ and retain all rates. Weight occupancy can only decrease. If $x_{u,g} = 0$, the activation cap forces $y_{u,g} = 0$. Otherwise the shared compute constraint gives $\tau_{u,g} y_{u,g} \le 1$, which is exactly the cap with a single copy. All remaining constraints are unchanged. □

Lemma A.4 (Zero-capacity pairs)**.** *If $B_{g,h} = 0$ and every modeled $S_{u\to v} > 0$, then all $f_{u\to v,g,h} = 0$. Routing variables on such pairs may be omitted.*

Proof. The bandwidth constraint is a sum of nonnegative terms bounded above by zero. Positive $S_{u\to v}$ forces every corresponding flow to vanish. □

Proposition A.5 (Sparse optimal extreme point)**.** *For a feasible fixed-placement LP, let q be its number of scalar equalities and inequalities, excluding variable nonnegativity. There is an optimal solution with at most q positive variables among (y, f, T).*

Proof. Introduce nonnegative slack variables and choose an optimal basic feasible solution. At most q variables, including slacks, are basic. Positive original variables are a subset of these. Such a solution exists because the feasible nonnegative polyhedron is nonempty and the objective is bounded above. □

This fact motivates sparse routing formulations. Candidate-pair screening reduces the feasible set; a matching certificate or an instance-level comparison establishes equality with the unrestricted optimum.

A.4 Profiles and memory reservations

Proposition A.6 (Monotone coefficient margins)**.** *Consider two instances of the same resource model with identical weights, budgets, and multiplicities. Suppose $\tau^{\text{true}}_{u,g} \le \overline{\tau}_{u,g}$, $a^{\text{true}}_u \le \overline{a}_u$, and $S^{\text{true}}_{u\to v} \le \overline{S}_{u\to v}$, all nonnegative with positive service times. Every solution feasible using the barred coefficients is feasible using the true coefficients, with rate cap $y_{u,g} \le x_{u,g}/\tau_{u,g}$ in both instances.*

Proof. Reducing the coefficients weakly decreases the compute, activation-memory, and bandwidth left-hand sides, and weakly increases the placement-dependent rate cap. The flow equalities do not depend on the perturbed coefficients. Thus every constraint remains satisfied. □

Coefficient margins quantify profiling uncertainty. Additional profiles, budgets, and constraints account for runtime effects.

Lemma A.7 (Buffers for the fluid schedule)**.** *Under Proposition F.6, choose period $H > 0$ and use a one-period lag at both compute and communication. A stationary schedule has at most $2c^{\text{in}}_v y_{v,h} H$ input token copies buffered at (v, h), and at most $2\alpha^{\text{out}}_{u\to v} y_{u,g} H$ output token copies buffered for edge $u \to v$ at g. The memory budget includes these buffers alongside in-service occupancy.*

Proof. At each period boundary, one input batch is available for computation and one output batch is available for communication. During the period, compute consumes the old input batch while links accumulate the next input batch; each has size $c_v^{\text{in}} y_{v,h} H$. Likewise, links consume the old output batch while compute produces the next output batch, each of size $\alpha_{u\to v}^{\text{out}} y_{u,g} H$. Their sums give the bounds for arbitrary relative ordering of the two independent resource schedules. Input bytes are bounded using the largest incoming token-copy size, or more tightly by maintaining separate edge buffers; output bytes use $S_{u\to v}$. □

KV residency and phase-specific profiles. For a fixed dense placement and sequence-affinity policy, let C_g bound concurrent sequences whose KV resides on GPU g, $L_{\max}$ bound their context length, and κ_g be the per-sequence KV bytes per context token on the layers resident there. A worst-case reservation is

$$\sum_u (1+\rho) W_u x_{u,g} + \sum_u a_u \tau_{u,g} y_{u,g} + \kappa_g C_g L_{\max} + Q_g \le M_g,$$

where Q_g reserves queue and runtime buffers. The constraint is linear when the residency policy and $C_g, L_{\max}, \kappa_g, Q_g$ are fixed. Selecting cache residency jointly with placement requires corresponding assignment variables and affinity constraints. Prefill and decode can use separate profiled service coefficients and rate variables while sharing weight placement and the GPU/link budgets. KV created during prefill and retained during decode is included once in the shared residency budget.

A.5 Latency and shared communication resources

Proposition A.8 (Utilization cap in an M/M/1 model)**.** *Approximate a queue by an M/M/1 system with fixed service rate $\mu > 0$, arrival rate $\lambda < \mu$, and queue waiting time W. For $t > 0$ and $0 < \delta < 1$, the sufficient condition*

$$\lambda \le \mu - \frac{\log(1/\delta)}{t}$$

implies $\Pr(W > t) \le \delta$. If the right-hand side is nonnegative and utilization is represented by $\sum_u \tau_{u,g} y_{u,g} = \lambda/\mu$, this yields the linear cap

$$\sum_u \tau_{u,g} y_{u,g} \le 1 - \frac{\log(1/\delta)}{\mu t}.$$

Proof. The M/M/1 waiting-time tail is $(\lambda/\mu)e^{-(\mu-\lambda)t}$, bounded above by $e^{-(\mu-\lambda)t}$. Substitution gives the probability bound; division by μ gives the utilization form. □

The utilization cap follows the stated M/M/1 assumptions, with μ obtained from a fixed profiled job mixture. Changes in the mixture or arrival/service distributions require corresponding queueing parameters.

Contention domains. Pairwise link budgets can be supplemented by shared NIC or fabric constraints. If domain d contains directed pairs L_d and has capacity C_d, add

$$\sum_{(g,h)\in L_d} \sum_{u\to v} S_{u\to v} f_{u\to v,g,h} \le C_d.$$

When these caps are necessary conditions of the physical fabric, they tighten the resource relaxation while retaining every physically feasible traffic pattern. Fabric scheduling is specified separately for the network topology.

Solver configuration. The reference planner uses `scipy.optimize.milp` with HiGHS (SciPy 1.14.1), a 20 s limit and relative MIP gap 0.002 for the original planning experiments. LP relaxation removes placement integrality. Presolve and thread settings follow backend defaults. Supplementary scalability studies use their stated limits and report the feasible objective and relevant upper bound separately. A feasible integer incumbent bounds the model optimum from below; an LP or branch-and-bound certificate bounds it from above.

B Group I: controlled simulation and planning details

This appendix details controlled topology emulation, mechanism analysis, and constraint ablations. The throughput comparison and physical-cluster results are in Section 5.

B.1 Methodology and Setup

B.1.1 Hardware Emulation Platform

We simulate the target topologies (C1–C6, Table 4) on an 8×NVIDIA H200 NVSwitch node by throttling three critical resources: injecting CUDA delay kernels for compute throughput, pre-allocating non-swappable dummy tensors to limit VRAM, and applying NCCL hooks to restrict pairwise bandwidth. These controls isolate compute, memory, and link bottlenecks on one physical platform.

B.1.2 Planner Implementation and Solver

The controlled study uses `scipy.optimize.milp` with a 20 s time limit and 2×10^{-3} relative MIP gap. Larger-instance studies use separate limits and report incumbents and bounds explicitly (Appendix E.3). Methods within each comparison receive the same hardware budgets. Their deployment templates determine the admissible placement and routing decisions.

B.1.3 Baselines and Metrics

Table 1 benchmarks ReVar against five baselines: a memory-aware *Greedy* heuristic, a randomized placement search, *LLM-PQ* [28] (rigid single-path pipelines), *HexGen-2* [10] (group-based disjoint pipelines), and *Helix* [16] (a dense-chain planner, inapplicable to the evaluated MoE graphs). The baselines implement these deployment templates in a common evaluation harness. The primary metric is steady-state throughput (tokens/s) in the controlled experiments. Workload-length and expert-activation statistics provide separate context for the sensitivity studies (Appendix C.3). For mechanism analysis, we also report per-GPU compute and peak-link bandwidth utilization.

Table 4: Cluster topologies used in evaluation.

ID	Composition	Interconnect	Scenario Type
C1	8 × A100 (80GB)	NVSwitch	Homogeneous (Standard)
C2	8 × H100 (80GB)	NVSwitch	Homogeneous (High-End)
C3	2 × [4 × A100]	NVLink + HDR200	Bandwidth Heterogeneity
C4	4 × H100 + 4 × A10	Mixed Bandwidth	Strong Heterogeneity
C5	1 × 24GB + 2 × 8GB	PCIe	Case Study (Memory)
C6	1 × 16GB + 2 × 7.45GB	PCIe	Case Study (Memory)

B.2 Main Results: Throughput

ReVar achieves the highest throughput in 14 of the 15 controlled settings and is within 0.2% of the best on the remaining one (Table 1). The largest gain over the strongest specialized planner is 15.8%. On homogeneous NVSwitch topologies (C1/C2), ReVar achieves similar throughput to specialized baselines. Under bandwidth heterogeneity (C3), ReVar successfully avoids inter-island bottlenecks by localizing high-volume traffic (Figure 8). When strong compute and bandwidth heterogeneity are combined (C4), ReVar's multi-path routing becomes critical, yielding the largest gains for MoE models with heavy cross-unit traffic. In the memory-constrained cases (C5/C6), feasibility depends on the deployment template. ReVar finds a feasible plan in each retained case, with its largest gain over Greedy on Mistral-7B/C5.

B.3 Mechanism Analysis

B.3.1 Mitigating stragglers via uneven replication and flow splitting.

To mitigate the *straggler effect* in heterogeneous clusters, ReVar balances load by assigning compute-heavy replicas and higher token volumes to fast GPUs while offloading lighter tasks to slower ones, thereby improving per-GPU utilization (Figure 7) and overall throughput (Table 1).

B.3.2 Protecting bottleneck links.

Figure 8 shows that ReVar explicitly keeps traffic away from constrained links. In C3 (two islands bridged by HDR200) and C4 (mixed interconnect), the optimized routing spreads activation flow across available high-bandwidth paths when possible and avoids driving the slowest links to saturation.

B.4 Ablation: Parallelism Adaptation

Table 5: Ablation on Llama-3-8B (C1). We vary the pipeline stage count S and sequence length L and report measured throughput of the resulting plans for Greedy/Random, ReVar, and constraint-relaxed variants. "No-BW" removes bandwidth constraints and "Temp-0" disables temporary-workspace accounting.

Configuration		**Baselines (tok/s)**		**ReVar Variants (tok/s)**		
S	L	**Greedy**	**Random**	**MILP-Full**	**MILP (No-BW)**	**MILP (Temp-0)**
2	512	13,631	13,596	**58,327**	58,186	58,181
2	1024	15,907	15,805	**66,570**	66,135	66,266
2	2048	15,162	15,089	**63,149**	63,111	63,061
4	512	23,965	13,615	**50,480**	50,462	49,589
4	1024	28,317	15,921	**58,877**	58,096	58,763
4	2048	27,661	15,127	**56,811**	56,784	56,164
8	512	38,198	23,820	**48,331**	39,405	44,705
8	1024	46,140	28,306	**56,560**	56,404	55,774
8	2048	46,688	27,486	54,766	**54,925**	54,638

Table 5 shows how bandwidth and workspace accounting affect the selected plans as stage count and sequence length change. Removing bandwidth constraints (NO-BW) is harmless for coarse pipelines but becomes noticeably worse when S is large (e.g., S=8, L=512): the solver over-fragments the plan, implicitly assuming free communication and thereby selecting placements that saturate links in practice. Disabling temporary-workspace accounting (TEMP-0) can also distort decisions at longer sequences by overestimating feasible parallelism under tight VRAM headroom.

B.5 Repeated-pod planning-time simulation

The scaling study uses independent eight-GPU C2-style H100/NVSwitch pods as the decomposition unit. It evaluates every complete-pod fleet size $n = 8q$, for $q = 1, \ldots, 12{,}500$. For each model, the eight-GPU component MILP is solved with HiGHS and a 60 s per-call limit; both component solves terminate optimally. The recorded component solver-call times are $t_8 = 0.001432625$ s for Llama-3-8B and $t_8 = 0.426257917$ s for Mixtral-8x7B. The serial planning-time estimate is $t_{\text{serial}}(8q) = q\, t_8$. The component programs contain 152 and 1,360 constraints, respectively, so the decomposed totals are $152q$ and $1{,}360q$. This construction measures planning cost for the repeated-pod topology shown in Figure 6.

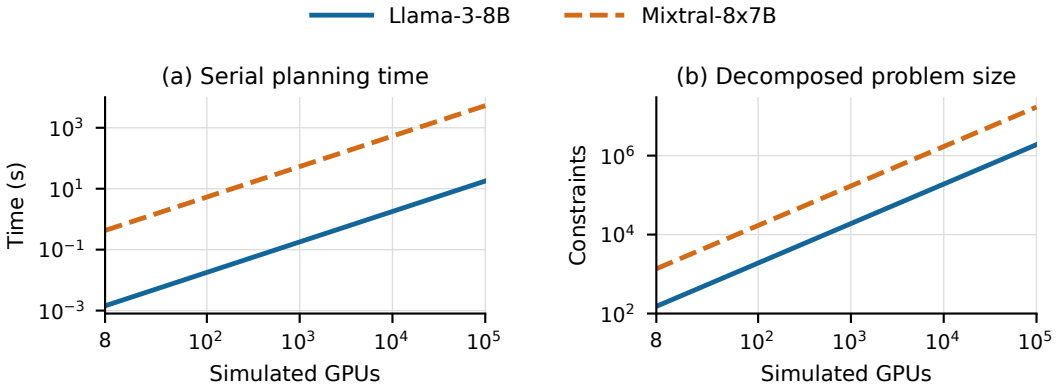

Figure 6: Planning-time simulation for repeated eight-GPU pods. Left: serial planning time assembled from measured per-pod solver calls. Right: total constraints in the decomposed program. The horizontal axis counts simulated GPUs; each plotted configuration consists of complete pods.

Table 6: Numerical milestones for the repeated-pod planning simulation. Time is the serial sum of measured eight-GPU component solver calls.

Simulated GPUs	Dense time (s)	MoE time (s)	Dense constraints	MoE constraints
8	0.00143	0.426	152	1,360
128	0.0229	6.820	2,432	21,760
1,024	0.183	54.561	19,456	174,080
10,000	1.791	532.822	190,000	1,700,000
100,000	17.908	5,328.224	1,900,000	17,000,000

C Supplementary experimental details

This appendix describes the unit graphs, solver settings, resource utilization, and workload profiles used alongside the main experiments.

C.1 Model graphs and solver settings

The graph constructor partitions each model into four layer groups. For dense models, embedding and output weights belong to the first and last stage, respectively, giving four units. An MoE group comprises one trunk, one dispatch unit, one unit per expert, and one combine unit; a shared output unit completes the graph. Consequently, the number of grouped expert units is four times the number of experts per layer, independently of the layer count. Table 7 gives these graph sizes and analytical weight footprints. The nine-node study uses the first four models with the same four-group graphs.

Table 7: Model and unit-graph specifications from the graph constructor. Weights use two bytes per parameter and decimal GB. The expert column gives experts per layer / selected experts per token. Graph counts use four layer groups.

Model	Layers	Experts / k	Weights (GB)	Expert units	All units	Edges
Llama-3-8B	32	–	16.06	0	4	3
Mistral-7B	32	–	14.48	0	4	3
Mixtral-8×7B	32	8 / 2	93.41	32	45	72
Mixtral-8×22B	56	8 / 2	281.24	32	45	72
DBRX	40	16 / 4	263.19	64	77	136

The C1–C6 graph-generation code reserves temporary memory equal to the weight footprint. The nine-node operator study instead uses measured maximum contiguous allocation budgets and reserves the graph's weights; its activation-workspace coefficient is zero. These settings apply to their

respective experiments. Sequence-resident KV storage is handled separately by the decode-capacity formulation.

For the nine-node study, the experimental environment uses SciPy 1.14.1 with HiGHS. Initial optimization calls use a 20 s limit and relative MIP tolerance 0.002. Random Best-of-3 uses NumPy's default generator with seed 0, followed by a fixed-placement LP for each of its three candidates. Recorded incumbents are subsequently improved by cross-evaluating compatible placements and by restricted and full-space searches; these refinement calls have separate time budgets. Upper bounds remain those of the unrestricted model. The 5% and 10% headroom factors scale measured compute, memory, pairwise-link, and shared-domain capacities together.

C.2 Resource utilization

Figures 7 and 8 visualize the allocation underlying the throughput comparisons. Compute utilization exposes idle capacity and uneven load across devices; incident-link utilization shows which communication boundaries carry the planned traffic.

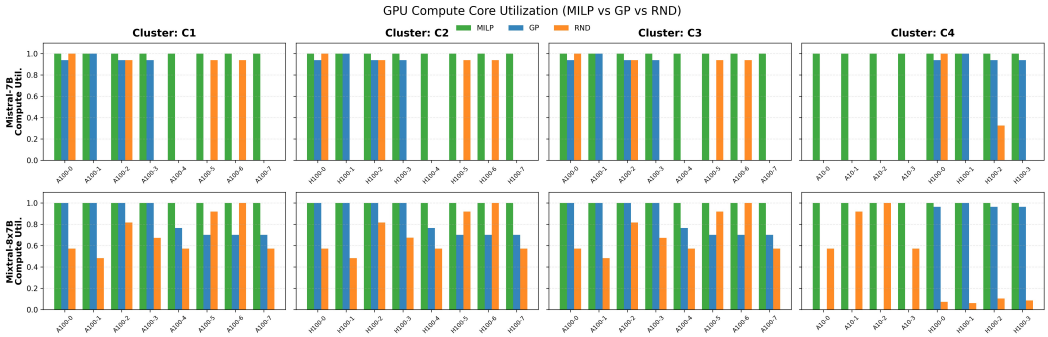

Figure 7: Per-GPU compute utilization across model graphs and the C1–C4 topologies. Each panel compares how the deployment methods distribute work across the available devices.

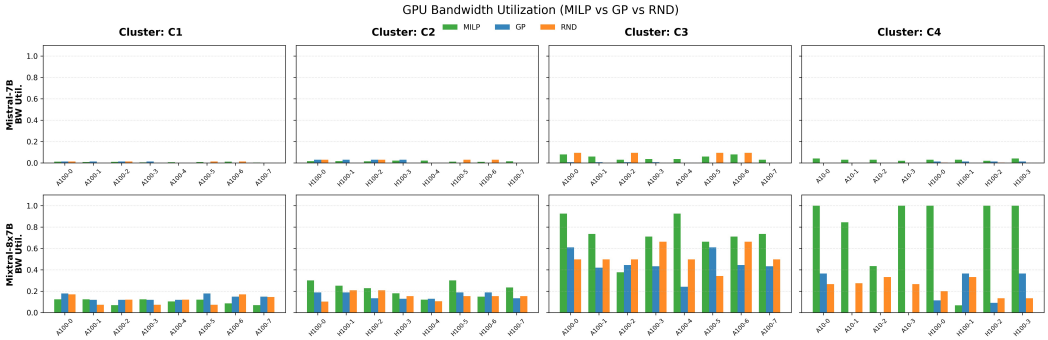

Figure 8: Maximum incident-link utilization for each GPU in C1–C4. Values range from zero for an idle link to one at the modeled capacity.

C.3 Workload profiles

The dialogue-corpus profiles characterize two sources of demand variation. Prompt and output lengths vary across requests (Figure 9), while expert selection varies across layers and corpora (Figure 10). These measurements motivate explicit expert-load coefficients and sequence-length-aware capacity accounting.

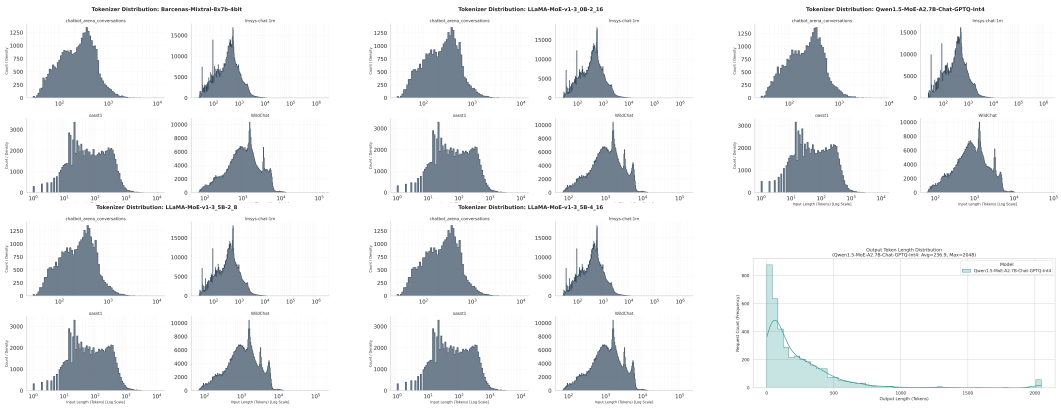

Figure 9: Prompt and output token-length distributions for the profiled MoE models. The final panel isolates the Qwen output-length tail under a 2,048-token truncation.

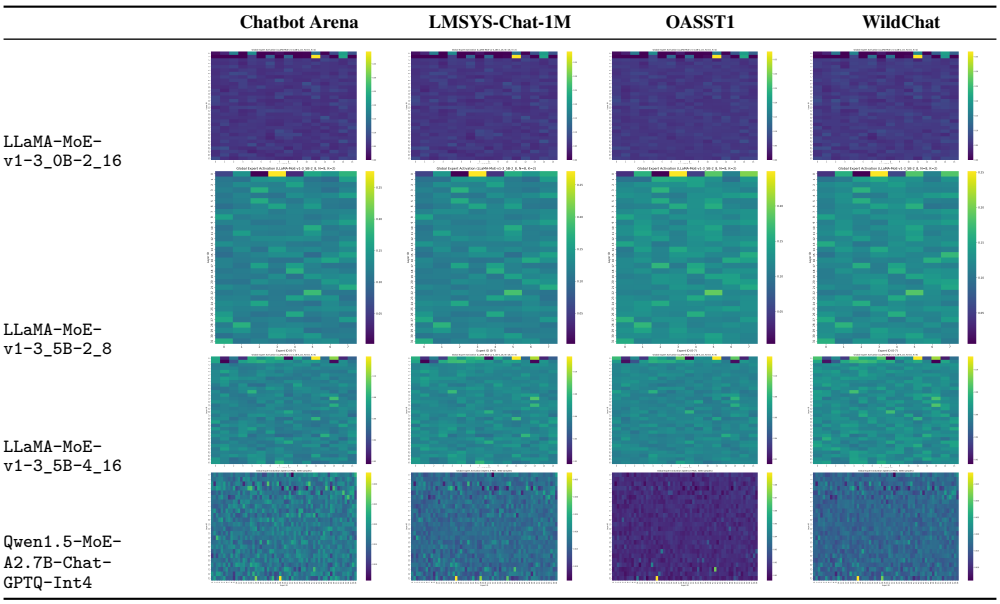

Figure 10: Expert activation profiles for four MoE models across four dialogue corpora. Each heatmap retains the layer and expert axes; colors show normalized activation frequency within the displayed profile.

D Physical-cluster protocols

D.1 Group II: L20/P4 execution protocol

The six-node experiment uses eight L20 GPUs and one P4 GPU. Its stage-level executor uses Gloo point-to-point communication on CPU tensors and models compute duration using GEMM-calibrated delays. Each stage has three warm-up iterations and eight timed iterations at 256 normalized tokens per step. If t_s is the mean execution time of stage s, the reported pipeline rate is $256/\max_s t_s$ and the latency is $\sum_s t_s$. Table 2 includes all four models and all 20 available model–method pairs in the statistical summary.

D.2 Group III: nine-node execution protocol

The nine-node cluster spans four availability zones and contains 38 GPUs: 12 A10, eight V100, eight P100, eight T4, and two P4 devices. We evaluate four model graphs with 5% and 10% resource headroom. Planning uses per-card, per-shape median GEMM throughput, maximum contiguous memory

allocation, and P10 bandwidth from intra-node CUDA copies and inter-node TCP measurements, all discounted by the selected headroom. Execution uses NCCL. Each deployment reserves its modeled weight footprint in GPU memory, executes FP16 cuBLAS GEMMs with the model's hidden and feed-forward dimensions, and transfers GPU-resident tensors through NCCL according to the planned routes. Each execution step represents 128 normalized input tokens; GEMMs use up to 256 rows. The workload combines shape-equivalent computation and communication. Appendix E.4 reports Transformer decoding with Attention and resident KV.

The underlying statistical summary contains 64 applicable, deployable candidate model–headroom–method entries; Table 3 reports deployment-aware selection and the template candidates. Identical placements, normalized execution rates, and normalized routes share one physical run, yielding 38 distinct deployments. Each deployment runs for at least 60 s after warm-up, producing 2,284 throughput windows and 72,530 GPU/host telemetry records in total. All 38 runs completed. The entries are means of the approximately one-second throughput windows.

Deployment selection. A shared, algorithm-identity-free calibrator scores candidate plans using compute fragmentation, message sizes, and topology features. Calibration leaves out the entire target model, fitting on the remaining models. The selected-candidate summary reports the highest measured throughput in all eight model–headroom groups. The ReVar row in Table 3 reports the measured throughput of the selected candidates. It chooses Full Replication for both Llama-3-8B headrooms and Mistral-7B at 5%, HexGen-2 for Mistral-7B at 10% and both Mixtral-8x7B headrooms, and unrestricted ReVar for both Mixtral-8x22B headrooms. A temporally separated validation reruns selected plans and the strongest historical references after reboot for 60 s. All eight selections meet its prespecified 3% regret criterion against these references.

Calibrator configuration. Two ExtraTrees regressors predict rank-level compute and communication times, using 2,000 trees each, minimum leaf size 2, all features at each split, and random seeds 0 and 1, respectively. Targets are natural logarithms of measured milliseconds, clipped below at 0.001 ms; predictions are exponentiated. GPU type is one-hot encoded, with unseen categories ignored. The compute regressor is fitted only on active ranks and receives the shape-profiled compute time, equivalent GEMM rows, chunk count at 256 rows, full-chunk count, tail rows, tail-presence indicator, hidden width, and feed-forward width. The communication regressor receives operation counts (total, at most 64 KiB, at most 16 KiB, intra-node, same-zone inter-node, and cross-zone), $\log(1+\text{bytes})$ for total/intra-node/same-zone/cross-zone traffic, and world size. Each fold excludes every runtime label from the target model, including both headrooms. For candidate c, the score is

$$\widehat{T}(c) = \frac{128 \times 1000}{\max_r \max(\widehat{t}_{c,r}^{\,\text{compute}}, \widehat{t}_{c,r}^{\,\text{comm}})},$$

with times in milliseconds and zero compute time for inactive ranks. Selection maximizes this score over the feasible candidates; ties use source-template name order.

E Additional validation of planning and execution

This section examines the effect of graph granularity, changes in expert demand, routing sparsity, and stateful decoding. The granularity, drift, and solver studies use the resource model; the decoding and request-control measurements execute a pretrained Transformer on one RTX 5060 Laptop GPU. A final planning study combines measured decode residence curves with heterogeneous resource budgets to examine sequence-affine KV allocation.

E.1 Unit-graph granularity

We partition the 32 layers of Mixtral-8x7B into equal-sized groups, retaining the same total weight and operation counts while varying the number of independently placeable units. Each group contains a dense trunk, dispatch and combine units, and eight expert units. Table 8 gives every tested granularity on C1–C4. Objectives are normalized within this controlled solver study to the corresponding one-group-per-layer graph. The four-group graph retains at least 99.9924% of that objective while using substantially fewer units. All instances close their integer optimality gaps to numerical tolerance.

Table 8: Mixtral-8x7B granularity sensitivity. C1–C4 report objective as a percentage of the respective 32-group optimum. The final column is the maximum wall time around the solver call across the four clusters. All 24 instances are included.

Groups	Units/edges	C1 (%)	C2 (%)	C3 (%)	C4 (%)	Max. time (s)
1	12/18	99.9924	99.9924	99.9924	99.9924	0.092
2	23/36	99.9924	99.9924	99.9924	99.9924	3.917
4	45/72	99.9924	100.0000	99.9924	100.0000	2.227
8	89/144	100.0000	100.0000	100.0000	100.0000	5.228
16	177/288	100.0000	100.0000	100.0000	100.0000	11.394
32	353/576	100.0000	100.0000	100.0000	100.0000	39.795

E.2 Adapting to changes in expert demand

For Mixtral-8x7B on an eight-A10 node with 10% memory headroom, we plan under one expert-demand distribution and evaluate the resulting placement under each of four distributions. The distributions comprise a uniform control and three saved activation traces, aggregated across layers. The original default trace is denoted A; the other two are WildChat and OASST1. We retain the placement and solve only the rate-and-routing LP for the new distribution. Regret is measured against a fresh integer plan for that distribution, each of which reaches zero reported gap.

For comparison, a fixed allocation is scaled by its most overloaded expert: if the planning and serving distributions are p^A and p^B, its sustainable rate is $T_A / \max_e(p_e^B / p_e^A)$. This gives the frozen-allocation entries in Table 9. Updating rates and routes absorbs most of the change without moving weights.

Table 9: Complete distribution-shift matrix. Each cell gives adaptive / frozen regret (%; lower is better) relative to the serving-distribution optimum. The fixed-placement LP takes 14.88 ms median (14.32–15.65 ms range).

Plan on / serve on	Uniform	Trace A	WildChat	OASST1
Uniform	0.000 / 0.000	0.000 / 2.520	0.000 / 4.385	0.000 / 2.371
Trace A	0.000 / 2.639	0.000 / 0.000	0.000 / 1.914	0.000 / 3.672
WildChat	0.000 / 3.383	0.000 / 1.525	0.000 / 0.000	0.106 / 4.509
OASST1	0.206 / 4.170	0.000 / 3.939	0.000 / 3.150	0.000 / 0.000

E.3 Sparse routing and integer certificates at 38 GPUs

We study Mixtral-8x7B on the measured nine-node, 38-GPU inventory with 10% memory headroom and the shared contention-domain constraints. Route screening retains self-links and the k highest-bandwidth remote destinations per source. A compact construction creates variables only for retained routes. With $k = 1$, its LP objective equals the unrestricted LP objective for this instance. Table 10 compares two implementations of this same screened LP: a dense tensor with excluded routes fixed to zero, and a compact variable array. Each measurement runs in a fresh process. The comparison isolates the cost of representing inactive variables.

Table 10: Identical top-1 screened LP, with dense or compact variable storage. Time and peak resident-set size (RSS) are medians of five independent processes. Both objectives equal the unrestricted LP value of 59,131.125 tok/s.

Construction	Total variables	Route variables	Time (s)	RSS (MiB)
Dense; inactive routes fixed	107,389	103,968	1.153	362.5
Compact; retained routes only	8,893	5,472	0.634	155.6

For integer planning, top-4 screening is combined with candidate placements appearing in either the screened LP solution or the strongest feasible template warm start. The resulting candidate set contains 623 of the 1,710 possible unit–GPU placements. Table 11 reports the feasible objective and

both relevant upper bounds. The branch-and-bound certificate applies to this screened candidate set. The unrestricted LP provides a separate bound for the full placement-and-routing space.

Table 11: 38-GPU screened integer solve. The branch-and-bound gap is $(U_{\mathrm{scr}} - T)/T$; the unrestricted-LP shortfall is $(U_{\mathrm{LP}} - T)/U_{\mathrm{LP}}$. Objectives are in tok/s. The checked maximum constraint residual is 3.68×10^{-10}.

Quantity	Value	Quantity	Value
Variables / binary variables	17,101 / 623	Feasible objective T	52,050.650
Retained route variables	13,680	Screened upper bound U_{scr}	52,570.941
Warm-start objective	19,720.431	Unrestricted LP U_{LP}	59,131.125
Solve time (s)	603.459	Screened gap (%)	1.00
		Unrestricted-LP shortfall (%)	11.97

E.4 Measured decode curves and sequence-affine KV planning

We measure SmolLM2-360M-Instruct in BF16 using Transformers on an RTX 5060 Laptop GPU with 8,151 MiB of physical memory. The test runs full Attention and cached autoregressive decoding. Synthetic token inputs match the prompt/output lengths of three Chatbot-Arena request classes, and each class/batch combination has three repetitions. Table 12 reports the complete factorial experiment. Decode throughput excludes prefill and counts the token steps represented by the recorded inter-token latencies (ITLs).

For each run, we compute decode residence R_c by summing its ITLs and the time-weighted context length $\bar{L}_c$ by weighting each cached context by its ITL. The materialized KV bytes agree exactly with the architecture-derived values in all runs. Reserving maximum context for the whole decode interval requires 28.98–33.61% more KV memory than this time-weighted occupancy.

Table 12: Measured single-GPU decode throughput (tok/s), median of three runs. All 36 runs complete; maximum throughput coefficient of variation is 3.80%.

Class	Prompt	Output	Batch 1	Batch 2	Batch 4	Batch 8
p50	248	247	29.28	58.55	115.07	235.04
p90	604	608	29.69	58.75	119.71	195.21
p99	1377	1358	29.63	60.28	90.74	101.24

KV residence and movement. A matched cache-policy experiment on the same model/GPU compares resident KV with a forced GPU–CPU–GPU cache transfer before every decode step, and with full-prefix recomputation at every step. All cases use batch one and generate 32 tokens. Table 13 includes every matched policy comparison, reporting decode rates separately from total request latency. Resident KV has the highest decode rate in all nine comparisons. At long contexts, prefill accounts for much of the request latency, so the total-latency ratio can conceal a substantial decode penalty.

All four CPU-transfer scenarios reproduce the resident output token for token. Table 13 reports output-token agreement for every policy.

Path/class formulation. A decode request is admitted to a dense-stage path p and retains that path until completion. Let λ_{pc} be the admission rate of class c on path p, and κ_s the KV bytes per context token at stage s. The mean KV occupancy and resident-sequence constraints on GPU g are

$$K_g = \sum_{p,c} \lambda_{pc} R_c \bar{L}_c \sum_{s:p_s=g} \kappa_s, \tag{8}$$

$$\sum_{p,c:g\in p} \lambda_{pc} R_c \le C_g. \tag{9}$$

The memory budget includes K_g alongside resident weights. For a maximum-context reservation, replace $\bar{L}_c$ by $L_c^{\max}$. A path can carry requests only when every required stage is placed; class shares,

Table 13: Complete matched KV-policy comparison. Resident and policy decode rates are in tok/s; Rate is policy/resident (%), and Latency is the total request-latency ratio, including prefill. Match is the percentage of generated tokens equal to the resident reference. Values are medians of three policy runs and five resident runs (three at prompt 8,160). The marked recomputation case has one completed run.

Policy	Prompt	Resident	Policy	Rate (%)	Latency (×)	Match (%)
CPU round trip	512	27.66	19.73	71.31	1.40	100.00
CPU round trip	2,048	27.94	13.00	46.51	1.80	100.00
CPU round trip	4,096	28.51	10.76	37.72	1.69	100.00
CPU round trip	8,160	26.61	7.82	29.38	1.00	100.00
Prefix recomputation	128	23.79	23.54	98.98	1.01	28.12
Prefix recomputation	512	27.66	18.76	67.81	1.46	100.00
Prefix recomputation	1,024	30.65	6.09	19.87	4.62	15.62
Prefix recomputation	2,048	27.94	1.93	6.92	10.56	100.00
Prefix recomputation	4,096†	28.51	0.0149	0.0523	801.24	100.00

GPU compute, and directed activation traffic add linear constraints. With profiled R_c, $\bar{L}_c$, and a fixed batch operating point C_g, this remains a MILP. The growing-KV variant represents mean occupancy under a stationary request mix; the maximum-context variant reserves for each sequence's largest cached context.

We apply this formulation to a four-stage Mistral-7B resource model with one each of A10, V100-32GB, T4, and P4, using measured device/link budgets. The request-class shares are 0.5/0.4/0.1. The SmolLM2 measurements supply a common residence/context sensitivity curve to all methods. Table 14 compares planner objectives under this shared empirical decode curve and the measured device/link budgets.

The baselines restrict the same path/class program: Helix places one stage per GPU, full replication uses complete model copies on fitting GPUs, and the greedy baseline fixes one complete copy on the largest-memory GPU. Each GPU's resident-sequence cap equals the selected batch operating point. Every method/scenario solve closes to zero reported gap. The full grid shows where ReVar recovers full replication and where component placement permits more resident paths.

Table 14: Complete KV-aware planner comparison (tok/s): two memory headrooms, four batch operating points, two occupancy models, and four methods (64 solves). Max reserves maximum context; Grow uses ITL-weighted context. These are planning objectives with the shared measured sensitivity curves described in text.

Headroom	Batch	KV	Helix	Full replication	Greedy	ReVar
5%	1	Max	29.63	88.89	29.63	88.89
5%	1	Grow	29.63	88.89	29.63	88.89
5%	2	Max	59.22	177.67	59.22	177.67
5%	2	Grow	59.22	177.67	59.22	177.67
5%	4	Max	109.41	306.57	109.41	328.24
5%	4	Grow	109.41	324.18	109.41	328.24
5%	8	Max	161.71	396.57	161.71	485.12
5%	8	Grow	161.71	408.51	161.71	485.12
10%	1	Max	29.63	59.26	29.63	88.89
10%	1	Grow	29.63	59.26	29.63	88.89
10%	2	Max	59.22	118.45	59.22	177.67
10%	2	Grow	59.22	118.45	59.22	177.67
10%	4	Max	109.41	218.82	109.41	328.24
10%	4	Grow	109.41	218.82	109.41	328.24
10%	8	Max	161.71	323.41	161.71	485.12
10%	8	Grow	161.71	323.41	161.71	485.12

E.5 Resident-KV request control

The request-control test combines weighted-deficit assignment, path pinning, per-path batches, and finite queue credits with real cached decoding. Two logical paths share the same RTX 5060 and model instance, with admission weights 1:2 and batch credits 2:4. The controller assigns requests once, then alternates between the two queues, executing each selected microbatch through completion. A queue defers admission to a batch when its pending requests exceed its credits. Each request uses a synthetic 64-token prompt and generates 16 tokens. Table 15 summarizes both complete runs. This tests request assignment and resident-state handling on one GPU, complementing the multi-GPU placement and communication measurements.

Table 15: Single-GPU integrated request-control test, two repetitions of the same 19 requests. Each run assigns 6/13 requests to the two paths. Output hashes match across runs.

Metric	Result
Requests completed / admitted	38 / 38
Maximum admission-share error (percentage points)	1.754
Backpressure events	10
Credit violations	0
Sequence path migrations	0
Reproducible output hashes across runs	2 / 2

F Formal statements and proofs

F.1 Model and formal statements

The statements below concern the profiled resource model. The unit graph is a finite DAG with a unique source and sink; every unit lies on a source-to-sink path. We assume $c_v^{\mathrm{in}} > 0$ for nonsource units, $\alpha_{u\to v}^{\mathrm{out}} \geq 0$, positive service times $\tau_{u,g}$, and nonnegative weights and activation coefficients. Activation edges have $S_{u\to v} > 0$. Multiplicities are normalized so that $m_{\mathrm{src}} = m_{\mathrm{snk}} = 1$. The closed-form compute laws specialize service times to F_u/P_g with $F_u, P_g > 0$.

Definition F.1 (Model tokens and token copies). A model token is one service-level throughput event. A token copy is an internal work item; branching may create several copies per model token. The rate $y_{u,g}$ counts output events of unit u on GPU g, and $f_{u\to v,g,h}$ counts copies transmitted along a logical edge between two replicas. Input and output multiplicities specify the aggregate copies consumed and emitted per event. Expected multiplicities are fixed coefficients for the profiled workload.

Definition F.2 (Multiplicity). The global work multiplicity m_u is defined by $m_{\mathrm{src}} = 1$ and $c_v^{\mathrm{in}} m_v = \sum_{u\to v} \alpha_{u\to v}^{\mathrm{out}} m_u$ for $v \neq \mathrm{src}$, as in Eq. (1).

Lemma F.3 (Unique multiplicity on a DAG). Under the graph assumptions above, the multiplicity equations have a unique nonnegative solution, computable in $O(|U| + |E|)$ arithmetic operations. **Proof:** Appendix F.3.1.

Resource constraints. Placement $x_{u,g} \in \mathbb{Z}_{\geq 0}$ activates execution at (u, g). The rate cap is $1/\tau_{u,g}$, equal to P_g/F_u under the FLOP-based specialization. The program maximizes T subject to

$$\sum_u (1+\rho) W_u x_{u,g} + \sum_u a_u \tau_{u,g} y_{u,g} \leq M_g, \quad \forall g, \tag{10a}$$

$$\sum_u \tau_{u,g} y_{u,g} \leq 1, \quad \forall g, \tag{10b}$$

$$y_{u,g} \leq x_{u,g}/\tau_{u,g}, \quad \forall u, g, \tag{10c}$$

$$\sum_h f_{u\to v,g,h} = \alpha^{\text{out}}_{u\to v} y_{u,g}, \quad \forall (u \to v), g, \tag{10d}$$

$$\sum_{u:u\to v} \sum_g f_{u\to v,g,h} = c^{\text{in}}_v y_{v,h}, \quad \forall v \neq \text{src}, h, \tag{10e}$$

$$\sum_{u\to v} S_{u\to v} f_{u\to v,g,h} \leq B_{g,h}, \quad \forall (g,h) : B_{g,h} < \infty, \tag{10f}$$

$$\sum_g y_{\text{src},g} = T, \qquad \sum_g y_{\text{snk},g} = T, \tag{10g}$$

$$x_{u,g} \in \mathbb{Z}_{\geq 0}, \quad y_{u,g} \geq 0, \quad f_{u\to v,g,h} \geq 0, \quad T \geq 0. \tag{10h}$$

The source receives exogenous work and has no node-input equation. Self-links have infinite capacity, so their bandwidth rows are omitted. The memory budget M_g excludes fixed system reservations; ρW_u is a chosen per-copy headroom allocation. The term involving a_u models in-service average occupancy. Explicit queue and KV reservations are described in Appendix A.

Lemma F.4 (In-service occupancy). Under the serial service abstraction, the time-average number of token copies actively in service at (u, g) is $y_{u,g}\tau_{u,g}$. If one active copy uses a_u bytes, its average in-service occupancy is $a_u \tau_{u,g} y_{u,g}$. Queueing and retained state require additional accounting. **Proof:** Appendix F.3.2.

Theorem F.5 (Resource-model throughput bound). Consider a stationary deployment whose fixed placement, average service times, per-replica output multiplicities, and input-copy requirements obey the stated model. Suppose its long-run compute and link use satisfy the budgets, and its memory accounting satisfies Eq. (10a). Then its sustained throughput is at most the optimum of Eqs. (10a)–(10h). **Proof:** Appendix F.3.3.

Proposition F.6 (Fluid time-sharing realization). Fix integer placement x and nonnegative (y, f, T) satisfying Eqs. (10b)–(10h). Assume inputs consumed by each unit are interchangeable divisible fluid, compute and directed links are independent preemptive resources, and queues have separately provisioned capacity. Then, after initialization with finite fluid inventory, a periodic schedule attains rates (y, f, T) while respecting the compute and link budgets. Request-level execution additionally requires operand identity at joins and request/KV affinity. **Proof:** Appendix F.3.4.

Proposition F.7 (Fixed-placement subproblem). For fixed x, optimizing over (y, f, T) is an LP. When its coefficients are rational, its optimum can be computed in polynomial time in the input encoding length. **Proof:** Appendix F.3.5.

Definition F.8 (ReVar). ReVar constructs a modular unit graph, profiles its resource coefficients, and optimizes placement, rates, and routing in the joint resource program. An integer solution provides a placement and aggregate rate plan. LP relaxation instead supplies a bound that must be distinguished from an integer deployment.

Proposition F.9 (Restricted deployment templates). The program contains single-stage-per-GPU pipeline templates, full model replication with on-device routing, and expert-parallel templates as restricted feasible sets, whenever their placements and rates meet the same budgets. **Proof:** Appendix F.3.6.

Definition F.10 (Helix-style chain restriction). The chain restriction uses a dense unit chain, binary placement with $\sum_u x_{u,g} \le 1$, and $a_u = 0$; all remaining resource and flow constraints are retained. The unit partition is held fixed in this template inspired by Helix.

Theorem F.11 (Strict containment of the chain restriction). The chain-restricted feasible set is a subset of the unrestricted resource program with the same graph and $a_u = 0$. There exist instances on which its best throughput is strictly smaller. **Proof:** Appendix F.3.7.

Theorem F.12 (Program size and NP-hardness). For $n = |U|$, $m = |E|$, and $p = |G|$, the full program has np integer placement variables, np continuous execution rates, mp^2 routing rates, and one throughput variable. Its scalar constraint count is $O(np + mp + p^2)$. Deciding whether throughput at least one is feasible is NP-hard, even on a dense chain with two GPUs, binary placement, weight-only memory, and nonbinding bandwidth. **Proof:** Appendix F.3.8.

F.2 Resource-dominated regimes

Unless explicitly stated otherwise, the regimes below use $a_u = 0$, $\rho = 0$, and $\tau_{u,g} = F_u/P_g$. Only positive-work units are included in ratio formulas. The optima refer to the stated resource model and layout restrictions.

Definition F.13 (Replica sets). $G[u] = \{g : x_{u,g} \ge 1\}$ and $r_u = |G[u]|$. A GPU's shared compute budget is counted once, irrespective of the number of colocated weight copies (Proposition A.3).

Definition F.14 (Per-token compute). $A_u = m_u F_u$ is the unit's compute demand per model token.

Definition F.15 (Per-token communication). $D_{u\to v} = S_{u\to v}\alpha^{\text{out}}_{u\to v} m_u$ is the logical edge's byte demand per model token.

Lemma F.16 (Global rate identity). Every feasible solution satisfies $Y_u = \sum_g y_{u,g} = m_u T$. **Proof:** Appendix F.3.9.

Theorem F.17 (Full model replication). If $\sum_u W_u \le M_g$ for every GPU, full replication with on-device routing achieves the resource-model optimum

$$T^\star = \frac{\sum_g P_g}{\sum_u A_u}.$$

No inter-GPU bandwidth assumption is needed. **Proof:** Appendix F.3.10.

Theorem F.18 (Compute-dominated stage replication). Consider $N \ge |U|$ identical GPUs with compute P, nonbinding bandwidth, and a dense chain with $A_u > 0$ and $W_u \le M$ for every unit. Require exactly one stage per GPU and at least one replica per stage. Then

$$T^\star = \max_{\substack{\sum_u r_u = N \\ r_u \ge 1 \text{ integer}}} \min_u \frac{r_u P}{A_u} \le \frac{NP}{\sum_u A_u}.$$

Equality holds if and only if all proportional counts $r_u = NA_u/\sum_w A_w$ are integers. For a fixed allocation, uniform splitting attains the minimum in the expression above. **Proof:** Appendix F.3.11.

Theorem F.19 (Compute-dominated expert replication). Let $\mathcal{E}_+$ be the positive-load experts of a top-k layer, with normalized frequencies $p_e > 0$, $\sum_e p_e = 1$, and $A_e = kp_e F_e$. Consider $R \ge |\mathcal{E}_+|$ dedicated homogeneous expert slots of compute P, each able to store any expert. Assume non-expert work has separately provisioned sufficient capacity and communication is nonbinding. With one expert per slot,

$$T^\star = \max_{\substack{\sum_e r_e = R \\ r_e \ge 1 \text{ integer}}} \min_e \frac{r_e P}{A_e} \le \frac{RP}{\sum_e A_e}.$$

The upper bound is attained exactly when $r_e = RA_e/\sum_{e'} A_{e'}$ is integral for every active expert. Equal F_e gives replica counts proportional to p_e in this integral case. **Proof:** Appendix F.3.12.

Theorem F.20 (Two-stage boundary). Consider a graph consisting of $a = \mathrm{src} \to b = \mathrm{snk}$, normalized multiplicities, and $D = D_{a\to b} > 0$. There are $N \geq 2$ homogeneous GPUs with compute P, weight capacity M, and capacity $B > 0$ on each inter-GPU directed link. Suppose $W_a, W_b \leq M$ and $W_a + W_b > M$. Restrict the layout to r GPUs hosting a and $N - r$ GPUs hosting b, with each GPU assigned one stage. Then

$$T[r] = \min\left\{\frac{rP}{A_a}, \frac{(N-r)P}{A_b}, \frac{r(N-r)B}{D}\right\}, \qquad T^\star = \max_{1\leq r\leq N-1} T[r].$$

Uniform routing over the complete bipartite boundary attains $T[r]$. **Proof:** Appendix F.3.13.

Corollary F.21 (Multipath gain at a bandwidth boundary). In Theorem F.20, suppose compute is large enough that bandwidth binds at the balanced split. Then $T^\star = \lfloor N^2/4 \rfloor B/D$. A deployment restricted to disjoint fixed GPU pairs has throughput at most $\lfloor N/2 \rfloor B/D$. The ratio of these bandwidth capacities grows as $\Theta(N)$. **Proof:** Appendix F.3.14.

Proposition F.22 (Computing the special cases). After computing multiplicities, the full-replication value and proportional fractional allocations take linear arithmetic time in the graph and GPU counts. Feasibility of target throughput t for the integer stage law is equivalent to

$$\sum_u \max\{1, \lceil tA_u/P \rceil\} \leq N.$$

The expert case has the same test with R slots. An exact integer optimum can be found by evaluating the finitely many thresholds jP/A_u, $1 \leq j \leq N$, or their expert counterparts. The two-stage optimum is found by scanning $N - 1$ splits. For a conservative dense-chain flow, Algorithm 1 uses at most $(L-1)|G|^2$ path extractions and polynomial time. **Proof:** Appendix F.3.15.

F.3 Proofs

F.3.1 Multiplicity uniqueness

The source value is fixed. Visit all remaining units in topological order and compute $m_v = (\sum_{u\to v} \alpha^{\mathrm{out}}_{u\to v} m_u)/c^{\mathrm{in}}_v$. All predecessors are already known, the denominator is positive, and the numerator is nonnegative. This proves existence, uniqueness, and nonnegativity by induction. Each edge is read once, giving the stated arithmetic complexity.

F.3.2 In-service occupancy

Over a long horizon of length H, the total service devoted to (u, g) is $\tau_{u,g} y_{u,g} H + o(H)$. Dividing by H gives its average active-service occupancy. Multiplying by a_u yields the average memory attached to service. Equivalently, Little's law applies to the service portion of the system with mean service time $\tau_{u,g}$. The total sojourn time also includes waiting, so this calculation does not upper-bound queued or retained memory.

F.3.3 Throughput bound

Take resident copy counts as x, long-run unit-event rates as y, and long-run logical-edge transmission rates as f. Stationarity and the assumed fixed per-replica multiplicities give edge-output and node-input balance. Source injection and sink completion both equal T. Average busy service time and transferred bytes give the compute and link inequalities. The memory inequality holds by the theorem's accounting assumption. An unplaced unit cannot execute; for a placed unit, shared compute gives $y_{u,g} \leq 1/\tau_{u,g} \leq x_{u,g}/\tau_{u,g}$. Thus all constraints hold for the observed throughput. Maximizing over this resource-feasible superset proves the bound.

F.3.4 Fluid schedule

Fix a period $H > 0$. On GPU g, reserve duration $H\tau_{u,g} y_{u,g}$ for each unit u. Their sum is at most H, so a preemptive serial schedule exists. On finite positive-capacity link (g, h), reserve duration $HS_{u\to v} f_{u\to v,g,h}/B_{g,h}$ for each commodity. The link budget bounds the sum by H. Zero-capacity pairs carry zero flow; infinite-capacity transfers need no service interval.

For every unit/edge/source-GPU triple, split its output batch according to $f_{u\to v,g,h}/(\alpha^{\text{out}}_{u\to v}y_{u,g})$ when the denominator is positive; otherwise all associated flows are zero. Edge balance makes these fractions sum to one. All rates here refer to divisible fluid, so fractional batch quantities are permitted.

Maintain two generations of buffers. During period k, GPU computation consumes only input fluid delivered by the end of period $k-1$; communication transmits only output fluid produced by the end of period $k-1$. Initialize each input buffer with one period of demand and each output buffer with one period of supply. Node balance supplies exactly the next period's total input demand; edge balance produces exactly the next period's output supply. Both buffers therefore regenerate at every period boundary, regardless of the ordering of compute and communication intervals. The two schedules use independent resources, so no additional serial-time requirement is introduced. Lemma A.7 bounds the finite stationary buffer inventory.

The source injects TH units and the sink completes TH units in every stationary period. Finite initialization does not affect the long-run rate. This proves the fluid schedule under the stated interchangeability and separate-buffer assumptions.

F.3.5 Fixed-placement LP

Fixing x makes every constraint affine in (y, f, T) and leaves a linear objective with continuous variables. Standard polynomial-time LP algorithms apply to rational input coefficients. Integrality and placement search are absent from this subproblem.

F.3.6 Restricted templates

For a dense chain, impose binary x and $\sum_u x_{u,g} \le 1$ to obtain the single-stage-per-GPU pipeline template; chain edges already connect consecutive stages. Full replication is obtained by fixing $x_{u,g} = 1$ for every unit and GPU and imposing $f_{u\to v,g,h} = 0$ for $g \neq h$. For an MoE graph, fix nonexpert placement on the chosen devices, require at most one expert per expert slot, and restrict edge routes to the dispatch/expert/gather pattern. Each construction fixes variables or adds linear restrictions to the same resource program, establishing feasible-set containment. It does not assert that an arbitrary imposed template fits every hardware budget.

F.3.7 Strictness of the chain restriction

Containment follows because the restriction adds constraints to the same program. For strictness, consider the following instance. Two GPUs each have $P_g = 1$ and $M_g = 2$, and both cross-GPU links have capacity 0.1. The dense chain $a \to b$ has $W_a = W_b = F_a = F_b = S_{a\to b} = 1$, unit multiplicities, and $a_u = \rho = 0$. Full replication fits on both GPUs. Each local full-model copy supplies $1/2$ token per second, giving total $T = 1$. Aggregate compute also bounds throughput by one, so this is the unrestricted optimum. Under one-stage-per-GPU placement, positive throughput requires one stage on each GPU. Every token traverses a cross-GPU link of capacity 0.1, so throughput is at most 0.1, which uniform service attains. The restricted optimum is therefore strictly smaller.

F.3.8 Size and NP-hardness

Counting variables gives $np + np + mp^2 + 1$. There are p memory rows, p compute rows, np activation-cap rows, mp edge-output rows, at most np node-input rows, at most p^2 bandwidth rows, and two throughput rows. Their total is $O(np + mp + p^2)$.

Reduce from PARTITION: positive integers $b_1, \dots, b_n$ sum to $2K$, and the question is whether some subset sums to K. Use two GPUs, each with memory K and compute n. Construct a dense chain of n units with $W_i = b_i$, $F_i = 1$, unit multiplicities, $a_i = 0$, and $\rho = 0$. Edge byte size is one and both directed link capacities are n, so they cannot bind at $T = 1$. Restrict placement to binary.

If a partition exists, place its units on one GPU and the remainder on the other. Set the hosted unit rates to one and route each chain edge between the appropriate hosts. Memory fits, each GPU processes at most n units per second, and every cross-link carries at most $n-1$ copies per second. Thus $T = 1$ is feasible.

Conversely, at $T \geq 1$, chain flow balance forces every unit's total rate to equal T. Each unit must have at least one copy. Their weights sum to $2K$, equal to total memory, so no unit can be duplicated and both GPUs must have weight exactly K. The units on one GPU form a partition. This polynomial construction proves NP-hardness of the restricted feasibility problem and hence of the general placement program.

F.3.9 Global rate identity

Sum each edge-output equation over source GPUs and each node-input equation over destination GPUs. This gives $c_v^{\text{in}} Y_v = \sum_{u \to v} \alpha_{u \to v}^{\text{out}} Y_u$ and $Y_{\text{src}} = T$. If $T > 0$, divide by T and invoke unique multiplicity. If $T = 0$, topological induction gives all $Y_u = 0$. Both cases yield $Y_u = m_u T$.

F.3.10 Full replication

Sum the compute inequalities after multiplying by P_g: $\sum_{u,g} F_u y_{u,g} \leq \sum_g P_g$. The global rate identity converts the left side to $T \sum_u A_u$, giving the upper bound. For attainment, place every unit on every GPU and set $T_g = P_g / \sum_u A_u$, $y_{u,g} = m_u T_g$. Route $f_{u \to v, g, g} = \alpha_{u \to v}^{\text{out}} y_{u,g}$ locally and set other flows to zero. Multiplicity balances give all input constraints; compute is saturated, weight-only memory fits, and no cross-link capacity is used. The total throughput matches the bound.

F.3.11 Integer stage replication

For fixed counts, stage u has total compute $r_u P$, while its demand is $A_u T$. Therefore $T \leq \min_u r_u P / A_u$. To attain this value, split stage rate $m_u T$ equally among its r_u replicas, and route each outgoing edge proportionally to the destination replica rates. In a chain, multiplicity balance gives the required inputs. Compute holds by the chosen minimum, weights fit, and bandwidth is nonbinding. Maximizing over integer counts gives the first equality.

Writing $t = \min_u r_u P / A_u$, we have $r_u \geq t A_u / P$. Summing gives $t \leq NP / \sum_u A_u$. Equality forces equality in every individual inequality, so each $r_u = N A_u / \sum_w A_w$ must be integral. Conversely, integral proportional counts attain the upper bound by the construction above.

F.3.12 Integer expert replication

Expert e requires $A_e T$ compute and receives $r_e P$ from its dedicated slots, giving $T \leq \min_e r_e P / A_e$. Splitting expert work equally over its slots attains this bound because nonexpert resources and communication are nonbinding by assumption. The same sum argument as for stages gives $T \leq RP / \sum_e A_e$ and shows that equality is equivalent to integer proportional counts. Expected top-k frequency gives $A_e = k p_e F_e$.

F.3.13 Two-stage boundary

For a fixed split, compute bounds throughput by rP/A_a and $(N-r)P/A_b$. All DT bytes per second must cross from the a hosts to the disjoint b hosts. There are $r(N-r)$ directed links, so bandwidth bounds throughput by $r(N-r)B/D$.

Set T to the minimum of these bounds. Let $y_{a,g} = m_a T / r$ on a hosts and $y_{b,h} = m_b T / (N-r)$ on b hosts. For each such pair set $f_{a \to b, g, h} = \alpha_{a \to b}^{\text{out}} m_a T / [r(N-r)]$. Every link then carries $DT / [r(N-r)] \leq B$. Edge-output balance holds by summing over h; node-input balance follows from $\alpha_{a \to b}^{\text{out}} m_a = c_b^{\text{in}} m_b$. Compute, weight memory, and activation caps hold. Thus the minimum is attained, and scanning the possible splits gives the optimum.

F.3.14 Multipath comparison

The product $r(N-r)$ is maximized by $r = \lfloor N/2 \rfloor$ or $\lceil N/2 \rceil$, with value $\lfloor N^2/4 \rfloor$. Under the stated compute assumption this gives the optimum. Disjoint fixed pairs use at most $\lfloor N/2 \rfloor$ directed crossing links, whose summed capacity is $\lfloor N/2 \rfloor B$. Dividing by D proves their upper bound and the asymptotic capacity ratio.

F.3.15 Explicit solution procedures

Multiplicity uses one topological pass. The full-replication value and fractional proportional counts require sums of A_u and P_g, so are linear-time arithmetic calculations. At target t, every stage needs at least $\max\{1, \lceil tA_u/P \rceil\}$ replicas. The target is feasible exactly when these requirements fit in N slots: any leftover slots can be assigned arbitrarily without lowering throughput. The objective of every integer allocation is one of the thresholds jP/A_u with $1 \le j \le N$, so evaluating the finite candidate set finds the exact optimum. This procedure is polynomial in the number of explicit GPUs and units; it does not claim a strongly polynomial bound for a compact binary encoding of N. The expert argument is identical. The two-stage formula is evaluated once per split.

For a conservative chain flow, a positive residual edge extends backward to a positive source path and forward to a positive sink path by node conservation. Each extraction makes at least one positive residual entry zero and preserves nonnegativity and conservation. At most $(L-1)|G|^2$ entries are initially positive, so there are no more than this many extractions. Scanning finite rows to extend each path gives polynomial work. This also proves Algorithm 1 terminates with an exact decomposition.

F.4 Dense-chain path decomposition

For a chain $u_1 \to \cdots \to u_L$, assume positive edge multiplicities and $L \ge 2$. Raw edge matrices are $F^i_{g,h} = f_{u_i \to u_{i+1}, g, h}$. Normalize them to model-token rates by

$$\widehat{F}^i_{g,h} = \frac{F^i_{g,h}}{\alpha^{\text{out}}_{u_i \to u_{i+1}} m_{u_i}}.$$

At an internal replica (u_i, g), normalized incoming and outgoing sums both equal $y_{u_i,g}/m_{u_i}$, so the matrices conserve flow. For unit multiplicities, normalization changes nothing. The extracted path rates sum to T; multiplying each edge's path rates by its normalizer recovers the original token-copy flows. Separate decompositions of MoE branches visualize their marginal routes but do not specify identity-matched joins.

Algorithm 1 Path decomposition of conservative chain flow

Input: normalized matrices $\widehat{F}^1, \ldots, \widehat{F}^{L-1}$ satisfying chain conservation
Set $R^i \leftarrow \widehat{F}^i$ for all i; initialize an empty path list
while some residual entry is positive **do**
  Choose g_1 with positive outgoing residual in R^1
  for $i = 1, \ldots, L-1$ **do**
    Choose g_{i+1} with $R^i_{g_i, g_{i+1}} > 0$
  end for
  Set $\delta = \min_{1 \le i < L} R^i_{g_i, g_{i+1}}$
  Subtract δ from every selected residual entry and record $(g_1, \ldots, g_L)$ with model-token rate δ
end while
Output: path list with total rate T

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: The abstract and introduction describe joint replica placement and routing for steady-state throughput planning. Section 5 separates resource-model experiments from the measured hardware and decode studies.

   Guidelines:

   - The answer [N/A] means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A [No] or [N/A] answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: Appendix A.1 discusses the steady-state abstraction, runtime effects, profiling dependence, and computational scaling.

   Guidelines:

   - The answer [N/A] means that the paper has no limitation while the answer [No] means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

   Answer: [Yes]

   Justification: Section 4 and Appendix F give the theorem assumptions and proofs.

   Guidelines:

   - The answer [N/A] means that the paper does not include theoretical results.
   - All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
   - All assumptions should be clearly stated or referenced in the statement of any theorems.
   - The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
   - Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
   - Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

   Answer: [Yes]

   Justification: Section 5 and Appendices B–E specify the algorithms, model graphs, solver settings, and measurement protocols.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - If the paper includes experiments, a [No] answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
   - If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
   - Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
   - While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
     (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
     (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
     (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).

(d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

   Answer: [No]

   Justification: An executable public code-and-data release is not provided.

   Guidelines:

   - The answer [N/A] means that paper does not include experiments requiring code.
   - Please see the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - While we encourage the release of code and data, we understand that this might not be possible, so [No] is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
   - The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
   - The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
   - At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
   - Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer) necessary to understand the results?

   Answer: [Yes]

   Justification: Section B.1 and Appendix C specify models, hardware, graph construction, and inference-planning settings.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [No]

   Justification: Repeated-measurement variability is reported; cross-cluster significance is not assessed.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The authors should answer [Yes] if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
   - The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
   - The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
   - The assumptions made should be given (e.g., Normally distributed errors).
   - It should be clear whether the error bar is the standard deviation or the standard error of the mean.
   - It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
   - For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g., negative error rates).
   - If error bars are reported in tables or plots, the authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: Section 5 and Appendices B–E report hardware, memory budgets, solver limits, and execution durations.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics https://neurips.cc/public/EthicsGuidelines?

   Answer: [Yes]

   Justification: Section 5 describes inference-deployment experiments on existing model configurations and workloads, with no new human-subject experiments or release of user data. Appendix A.1 discusses limitations and broader impacts.

   Guidelines:

   - The answer [N/A] means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer [No], they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [Yes]

    Justification: Appendix A.1 discusses access and hardware-utilization benefits alongside increased resource demand and reduced deployment costs for harmful applications.

    Guidelines:

    - The answer [N/A] means that there is no societal impact of the work performed.
    - If the authors answer [N/A] or [No], they should explain why their work has no societal impact or why the paper does not address societal impact.
    - Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
    - The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate Deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
    - The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
    - If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pre-trained language models, image generators, or scraped datasets)?

    Answer: [N/A]

    Justification: This paper does not release pretrained models, scraped datasets, or other assets requiring controlled access for misuse prevention.

    Guidelines:

    - The answer [N/A] means that the paper poses no such risks.

- Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
- Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
- We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer: [No]

    Justification: Sources are cited; a complete license inventory is not supplied.

    Guidelines:

    - The answer [N/A] means that the paper does not use existing assets.
    - The authors should cite the original paper that produced the code package or dataset.
    - The authors should state which version of the asset is used and, if possible, include a URL.
    - The name of the license (e.g., CC-BY 4.0) should be included for each asset.
    - For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
    - If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
    - For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
    - If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [N/A]

    Justification: The paper does not accompany a public release of a new dataset, pretrained model, or software package.

    Guidelines:

    - The answer [N/A] means that the paper does not release new assets.
    - Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
    - The paper should discuss whether and how consent was obtained from people whose asset is used.
    - At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [N/A]

    Justification: The study does not involve crowdsourcing or experiments with human participants.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
    - According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [N/A]

    Justification: The study does not involve human participants and does not report a human-subject study requiring IRB review.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
    - We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
    - For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

    Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does *not* impact the core methodology, scientific rigor, or originality of the research, declaration is not required.

    Answer: [N/A]

    Justification: LLMs are the inference workloads being deployed; LLM-generated judgments or outputs are not an original component of the planning method.

    Guidelines:

    - The answer [N/A] means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
    - Please refer to our LLM policy in the NeurIPS handbook for what should or should not be described.