International Journal of Scientific Research in Engineering and Management (IJSREM)
Volume: 10 Issue: 10 | Oct - 2026 SJIF Rating: 8.659 ISSN: 2582-3930

Communication-Aware Scheduling for Distributed LLM Inference on Heterogeneous Edge and Consumer Devices: A Survey

Prof. Maheshwari Bhalshankar, Darshan Prakash Vadawane, Prathmesh Sanjay Manwar, Omkar Vivekanand Kinage, Sarthak Appasaheb Thanekar
Department of Artificial Intelligence and Machine Learning, Zeal Polytechnic, Pune, Maharashtra, India

Abstract—Large language model (LLM) inference can exceed the memory and compute capacity of a single consumer device. Distributed inference pools resources across machines, but performance depends on communication latency, bandwidth, hardware heterogeneity, availability, and reconfiguration cost. This survey reviews collaborative inference, heterogeneous scheduling, prefill–decode disaggregation, and memory offloading. Petals, HexGen, HexGen-2, MoLink, DistServe, Splitwise, and FlexGen are compared qualitatively because their workloads and evaluation settings differ. We then outline a conceptual adaptive execution-plan selector that profiles devices and links, filters infeasible layer partitions, estimates execution cost, and accounts for interruption risk and replanning overhead. The framework is a research proposal, not a validated system.

Keywords—Large language models, distributed inference, heterogeneous devices, communication-aware scheduling, pipeline parallelism, edge computing.

I. INTRODUCTION

Large language models support generation, summarization, question answering, and coding, but their inference cost depends on model size, precision, context length, and KV-cache memory. When one machine cannot hold or efficiently execute a model, options include quantization, offloading, model partitioning, or hosted inference; each trades latency, cost, privacy, and complexity.

Distributed inference divides model computation across devices. In a layer-wise pipeline, each device executes a contiguous group of transformer layers and transfers intermediate activations to the next device. This can combine memory and compute capacity, but end-to-end latency may be limited by the slowest stage or network link; disconnections can interrupt requests.

Consumer and edge devices differ in GPU generation, CPU capability, memory, operating system, and network quality. Equal layer counts therefore need not balance execution time, and a fast device connected by a slow link may worsen latency.

Representative systems address different parts of this problem. Petals enables collaborative inference across participant-hosted model blocks [1], [2]; HexGen and HexGen-2 schedule heterogeneous execution and disaggregated inference [3], [4]; Edge-LLM studies cost-aware layer allocation [5]; MoLink targets communication scheduling on consumer GPUs [6]. DistServe and Splitwise separate prefill and decoding [7], [8], while FlexGen optimizes memory offloading [9].

This survey compares these approaches without assuming that one scheduler replaces them all. Its contributions are a concise architecture taxonomy, qualitative comparison of system objectives and trade-offs, and a conceptual adaptive selector for small heterogeneous PC deployments.

The key question is when distributing computation is worthwhile after accounting for communication, memory, device availability, and the cost of changing plans.

II. BACKGROUND AND PROBLEM FORMULATION

Autoregressive inference has two phases: prefill processes the prompt and builds the attention key-value (KV) cache; decoding generates tokens iteratively using that cache. Prefill and decoding have different compute, memory, and communication patterns.

Layer-wise pipelines transfer activations between layer groups, whereas disaggregated serving may transfer KV-cache state between prefill and decode workers. These architectures solve related but distinct problems.

Pipeline parallelism assigns contiguous transformer-layer groups to different devices. For L layers and N devices, a plan chooses layer boundaries and must respect each device's compute and memory limits. Tensor parallelism splits operations within a layer and usually requires more frequent communication; hybrid methods combine both.

HexGen shows the value of asymmetric allocations for heterogeneous GPUs and networks [3]. On weak or variable consumer networks, communication-intensive tensor parallelism may be less attractive than a simpler layer-wise pipeline.

Let D = {d1, …, dN} denote available devices. Each device has compute throughput, free memory, current load, and an availability estimate; each link has effective bandwidth Bij and latency Lij. A candidate plan m specifies participating devices, layer boundaries, and transfer paths.

A plan is feasible only when assigned weights, runtime buffers, and required KV-cache state fit available memory. A first-order latency estimate is the sum of compute, communication, and coordination time; it must be calibrated because overlap and batching can change the critical path.

$$\hat{T}(m) = \hat{T}\text{compute}(m) + \hat{T}\text{communication}(m) + \hat{T}\text{coordination}(m) \quad (1)$$

This equation is an approximation, not an exact performance model; measurements are required to calibrate it.

Given a prompt, model, and time-varying heterogeneous devices, select a feasible execution plan that minimizes expected latency while considering communication, interruption risk, and reconfiguration cost. Additional devices should be avoided when coordination and transfer costs outweigh their compute or memory benefit.

This is a structured narrative survey of selected studies on distributed LLM execution, heterogeneous resources, communication overhead, scheduling, and memory management.

The studies are compared by architecture, distribution strategy, optimization target, and limitations, covering collaborative inference, pipeline/tensor parallelism, prefill–decode disaggregation, and memory offloading.

The synthesis motivates a conceptual adaptive execution-plan selector. This survey is not exhaustive, and performance

results are interpreted within each paper's experimental setting rather than ranked directly.

IV. TAXONOMY OF DISTRIBUTED INFERENCE ARCHITECTURES

Petals distributes model blocks across participant-operated servers so clients can use models without storing all weights locally [1], [2]. Its relevance is collaborative access and operation despite changing participation; availability, trust, and network variability remain challenges.

HexGen uses asymmetric pipeline and tensor parallelism to allocate work according to GPU and network differences [3]. HexGen-2 extends heterogeneous scheduling to prefill–decode disaggregation and KV-cache communication [4]. Equal layer counts are not necessarily balanced across devices.

DistServe optimizes placement and parallelism for goodput under service-level objectives [7], while Splitwise separates prefill and decoding to match phase-specific workloads to suitable hardware [8]. These approaches primarily target serving systems rather than arbitrary peer-to-peer layer partitioning.

FlexGen supports generation with limited GPU memory by moving tensors across GPU, CPU memory, and disk [9]. Offloading moves data through a memory hierarchy, whereas distributed inference transfers activations between independent machines; a hybrid design must model both costs.

For a small prototype, layer-wise pipeline partitioning is a practical starting point because transfer boundaries are explicit. Tensor parallelism is more suitable when interconnects are fast; phase disaggregation targets multi-request serving, while offloading primarily addresses memory capacity.

V. COMMUNICATION MECHANISMS AND OPTIMIZATIONS

At each pipeline boundary, a device sends the hidden-state activation to the next partition. Transfer size depends on sequence length, hidden dimension, batch size, and precision; repeated small transfers can be latency-sensitive even when bandwidth is adequate.

A first-order transfer estimate is $T_{transfer} \approx L + S/B$, where S is payload size, B effective bandwidth, and L fixed link latency. This omits congestion and protocol overhead but explains why both bandwidth and latency matter.

MoLink chunks and schedules large prefill transfers to reduce interference with latency-sensitive decode traffic, reporting up to 46% reductions in TTFT, TPOT, and latency in its evaluated settings [6]. This result should not be assumed for every peer-to-peer pipeline.

Activation compression can reduce traffic but adds processing overhead and may affect numerical quality. Weight quantization reduces model memory but does not directly eliminate activation-transfer latency; end-to-end benefits must be measured.

KV-cache state may need to move when prefill and decoding run on different workers, and transfer volume can grow with context length. A first layer-wise prototype should keep cache ownership clear and defer general cache migration.

Placement should account for compute and link cost together. Measure bandwidth and latency under realistic load, because Wi-Fi interference and background traffic can change the best partition.

If a worker disconnects, the coordinator must detect failure and decide whether to resume, recompute, or restart. Recovery can require weight loading or cache restoration; the prototype should distinguish these behaviors rather than claim transparent recovery.

VI. TASK AND RESOURCE SCHEDULING

Static scheduling fixes layer boundaries before inference and is easy to reproduce, but may perform poorly when load or network conditions change. Dynamic scheduling can adapt, yet profiling and reconfiguration add overhead, so replanning should require a meaningful expected benefit.

Compute-only scheduling balances measured execution time but can choose partitions with expensive transfers. Equal layer counts are a weak proxy for equal compute because layer costs, utilization, memory pressure, and hardware differ.

Communication-aware scheduling estimates compute and transfer costs together. HexGen and HexGen-2 incorporate heterogeneous resources and communication [3], [4]; MoLink shows that transfer order and chunk size can also affect latency [6].

Consumer devices may sleep, disconnect, or become busy with their owners' applications. The scheduler should use recent health and availability observations and include the cost of loading weights, moving state, restarting the pipeline, and losing work.

For each feasible plan m, minimize the objective:

$J(m) = \hat{T}(m) + \lambda_r R(m) + \lambda_c C(m)$, where $\hat{T}(m)$ is predicted inference latency, $R(m)$ is normalized interruption risk, and $C(m)$ is reconfiguration cost. The weights λ_r and λ_c control the importance of risk and switching overhead.

Memory and device compatibility are hard constraints. The scheduler selects the feasible plan with the lowest estimated objective and updates estimates when load, network conditions, or device availability change.

VII. COMPARATIVE ANALYSIS OF EXISTING APPROACHES

Petals emphasizes collaborative model hosting [1], [2]; HexGen emphasizes heterogeneous parallelism [3]; HexGen-2 combines heterogeneity with phase disaggregation and KV-cache transfer [4]. MoLink targets communication scheduling on consumer GPUs [6], DistServe and Splitwise focus on phase separation [7], [8], and FlexGen targets memory offloading [9].

Table I. Comparative Analysis of Distributed LLM Inference Systems

System	Primary approach	Main optimization target	Communication / resource consideration	Limitation relevant to this survey
Petals	Collaborative inference across distributed machines	Running large models using resources contributed by multiple machines	Depends on communication between distributed model blocks and available peers	Variable network conditions and peer availability can affect performance
HexGen	Asymmetric pipeline and tensor parallelism	Efficient inference across heterogeneous GPUs	Considers differences in GPU capability and network connectivity	Performance depends on accurate placement and suitable communication paths
HexGen-2	Heterogeneous inference with prefill–decode disaggregation	Serving performance through resource allocation and scheduling across inference phases	Must account for KV-cache transfer and coordination between phases	Disaggregation introduces additional communication and coordination requirements
MoLink	Distributed inference using consumer-grade GPUs	Coordinating computation and communication over weaker networks	Focuses on transmission scheduling and communication efficiency	Network variability and differences between consumer devices remain challenging
DistServe	Disaggregated prefill and decode serving	Reducing interference between prompt processing and token generation	Requires resource allocation and movement of KV-cache data	Benefits depend on the workload, available resources, and transfer overhead
Splitwise	Separating prefill and decode across different hardware resources	Matching different inference phases to suitable hardware	Requires coordination and appropriate resource allocation across phases	Depends on hardware availability and the overhead of separating the phases
FlexGen	Memory-aware inference with GPU, CPU, and disk offloading	Enabling inference with limited GPU memory	Trades data movement and offloading for lower GPU-memory requirements	Data-transfer overhead can limit performance

Note: This is a qualitative comparison of principal design choices, not a performance ranking. Limitations describe implications relevant to this survey.

The systems address different bottlenecks rather than providing one universal solution. Distributed execution can make larger models accessible, but communication can reduce its benefit; phase disaggregation enables resource specialization but adds KV-cache transfer; offloading reduces GPU-memory demand but increases data movement.

The appropriate approach depends on the model, workload, hardware, network, and memory budget. A fair scheduler comparison must use the same model, prompts, hardware, network conditions, and measurement procedure.

Published results illustrate different objectives: HexGen reports up to 2.3× lower latency in a deadline-oriented comparison or up to 4× higher traffic rates than a homogeneous baseline under its tested budget [3]. HexGen-2 reports up to 2× throughput improvement and 1.5× lower average latency in its evaluation [4]; MoLink reports up to 46% reductions in selected latency metrics [6].

These figures are not directly comparable because models, hardware, networks, baselines, workloads, and objectives differ. They should not be combined into a single ranking.

The literature treats heterogeneity, communication, reliability, and memory as interacting concerns. The narrower open question is whether a lightweight runtime selector can choose useful layer partitions for small consumer-PC deployments under changing conditions without excessive planning overhead.

Prior systems do not simply ignore communication or heterogeneity; they optimize different combinations of these factors. A new framework must be evaluated against existing methods rather than claiming that communication-aware scheduling is itself new.

VIII. OPEN CHALLENGES AND RESEARCH DIRECTIONS

Profiling must be lightweight because continuous measurement of every device pair and partition adds overhead. Combine periodic measurements with runtime observations and avoid replanning when estimated differences are smaller than measurement noise.

Compute and transfer costs vary with model architecture, precision, batch size, prompt length, and generation length. Evaluation should include varied workloads and separate calibration data from test cases.

Long contexts increase KV-cache memory and can change the bottleneck. A scheduler must respect cache ownership or explicitly model migration cost.

Consumer devices are not always available and must be able to reclaim resources. Controlled failures, resource limits, privacy, and model-weight access should be considered in evaluation.

Reproducible experiments should report model and quantization, prompt and output lengths, device specifications, memory, network type and measurements, software versions, and repeated-run variability.

IX. PROPOSED SOLUTION: ADAPTIVE EXECUTION-PLAN SELECTION

The proposed framework selects among feasible layer-wise plans for heterogeneous consumer PCs. It combines compute and communication estimates with device availability and reconfiguration overhead; it does not claim to introduce a new form of model parallelism.

1) Receive the request and constraints; 2) discover devices and collect free-memory, load, and health information; 3) measure candidate links; 4) generate a limited set of layer partitions; 5) reject plans that violate memory or compatibility constraints; 6) estimate compute, transfer, interruption, and switching costs; 7) execute the lowest-cost feasible plan; and 8) monitor performance and replan only when conditions change enough to justify it.

Modules comprise a device profiler, network monitor, candidate planner, cost estimator, scheduler, runtime executor, and monitor/recovery manager. Together they measure resources, generate feasible contiguous layer assignments, execute activation transfers, and decide whether to continue or replan.

The initial algorithm uses bounded enumeration: generate a small set of contiguous layer partitions, reject infeasible assignments, estimate latency and risk, add reconfiguration penalties, and select the feasible plan with the lowest objective. During execution, update estimates and replan only after sustained degradation or failure.

Input: model M, prompt q, available devices D, current plan m_cur.

1. Profile devices and candidate links; generate candidate plans.

2. Reject plans that violate memory or compatibility constraints.

3. Estimate $\hat{T}(m)$, R(m), and C(m) for each feasible plan.

4. Select $m^* = \text{argmin } J(m)$; execute and collect runtime measurements.

5. If failure or sustained degradation occurs, refresh profiles and replan.

Output: generated response and execution metrics.

The first prototype should support a decoder-only model, layer-wise pipeline partitioning, two or three devices, and a small candidate set. Defer arbitrary tensor parallelism, multi-replica serving, and general KV-cache migration until the baseline is repeatable.

The intended contribution is an evaluated framework and reproducible comparison—not a claim that communication-aware scheduling is new. Novelty must be established by comparing the specific combination of low-overhead profiling, layer-wise planning, availability, and reconfiguration-cost control with prior work.

Evaluate against equal-layer static partitioning, compute-only partitioning, and a static communication-aware partition; include local execution or offloading if feasible. Keep model, quantization, prompts, generation settings, and hardware consistent across baselines.

Measure TTFT, TPOT, total latency, throughput, activation-transfer time, per-device peak memory, plan-selection overhead, and recovery time. Vary network conditions, device count, background load, prompt length, and output length; report repeated-run medians and variability.

The hypothesis is that communication-aware selection helps under some heterogeneous network conditions but may add overhead when links are fast or the model fits on one device. This is a hypothesis to test, not a result.

X. CONCLUSION

Distributed LLM inference can pool memory and compute across devices, but introduces communication, scheduling, and reliability costs. Existing systems explore collaborative hosting, heterogeneous parallelism, phase disaggregation, communication scheduling, and memory offloading; their results cannot be ranked directly across different evaluations.

This survey proposes a lightweight layer-wise execution-plan selector based on compute time, communication cost, availability, and reconfiguration overhead. Implementation and controlled experiments are required to validate its cost model and determine when adaptation improves performance.

ACKNOWLEDGMENT

The authors thank Prof. Maheshwari Bhalshankar for project guidance, Prof. Vaishnavi Patharwat for departmental support, Prof. Vijay Mohite for institutional support, and Zeal Polytechnic for providing the academic environment for this work.

REFERENCES

[1] A. Borzunov, D. Baranchuk, T. Dettmers, M. Ryabinin, Y. Belkada, A. Chumachenko, P. Samygin, and C. Raffel, –Petals: Collaborative Inference and Fine-tuning of Large Models,– in Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics, Volume 3: System Demonstrations, pp. 558–568, 2023. doi: 10.18653/v1/2023.acl-demo.54. Available: https://aclanthology.org/2023.acl-demo.54/

[2] A. Borzunov, M. Ryabinin, A. Chumachenko, D. Baranchuk, T. Dettmers, Y. Belkada, P. Samygin, and C. Raffel, "Distributed Inference and Fine-tuning of Large Language Models Over The Internet," Advances in Neural Information Processing Systems 36, 2023. doi: 10.52202/075280-0540. Available: https://papers.nips.cc/paper_files/paper/2023/hash/28bf1419b9a1f908c15f6195f58cb865-Abstract-Conference.html

[3] Y. Jiang, R. Yan, X. Yao, Y. Zhou, B. Chen, and B. Yuan, –HexGen: Generative Inference of Large Language Model over Heterogeneous Environment,– in Proceedings of the 41st International Conference on Machine Learning, PMLR 235, pp. 21946–21961, 2024. Available: https://proceedings.mlr.press/v235/jiang24f.html

[4] Y. Jiang, R. Yan, and B. Yuan, –HexGen-2: Disaggregated Generative Inference of LLMs in Heterogeneous Environment,– in International Conference on Learning Representations, 2025. Available: https://proceedings.iclr.cc/paper_files/paper/2025/hash/0b941a1e5fbce23fe46b049999d04ed0-Abstract-Conference.html

[5] A. Habibi and E. Erçetin, –Edge-LLM Inference With Cost-Aware Layer Allocation and Adaptive Scheduling,– IEEE Access, 2025. doi: 10.1109/ACCESS.2025.3592308. Available: https://doi.org/10.1109/ACCESS.2025.3592308

[6] L. Jin, K. Zhang, Y. Chen, Y. Zhuo, R. Li, Y. Gao, B. Yang, Z. Cai, and W. Dong, –Distributed LLM Serving on Consumer-Grade GPUs by Reconciling Computation and Communication,– in Findings of the Association for Computational Linguistics: EMNLP 2025, pp. 17633–17642, 2025. doi:

10.18653/v1/2025.findings-emnlp.957. Available: https://aclanthology.org/2025.findings-emnlp.957/

[7] Y. Zhong et al., –DistServe: Disaggregating Prefill and Decoding for Goodput-optimized Large Language Model Serving,– in 18th USENIX Symposium on Operating Systems Design and Implementation (OSDI), 2024. Available: https://www.usenix.org/conference/osdi24/presentation/zhong-yinmin

[8] A. Patel et al., –Splitwise: Efficient Generative LLM Inference Using Phase Splitting,– in Proceedings of the 51st Annual International Symposium on Computer Architecture (ISCA), 2024. doi: 10.1109/ISCA59077.2024.00019. Available: https://doi.org/10.1109/ISCA59077.2024.00019

[9] W. Sheng et al., –FlexGen: High-Throughput Generative Inference of Large Language Models with a Single GPU,– in Proceedings of the 40th International Conference on Machine Learning, PMLR 202, 2023. Available: https://proceedings.mlr.press/v202/sheng23a.html