TAILOR: Token-Aware Partitioning and Routing for Edge–Cloud Transformer Inference

Xiaoyao Huang*, Remington R. Liu*, Jie Wu*†
*China Telecom Cloud Computing Research Institute
†Temple University
Email: huangxy32@chinatelecom.cn, remington.liu@outlook.com, jiewu@temple.edu

***Abstract*—Large language models (LLMs) are increasingly deployed in collaborative edge–cloud environments to reduce latency and cloud cost. However, autoregressive transformer inference exhibits strong token-level variability: output lengths are unknown at request arrival, KV-cache memory grows during decoding, and long-tailed requests can trigger out-of-memory (OOM) failures on memory-constrained edge devices. Existing static partitioning methods collapse this token-dependent cost surface into a single layer-to-device mapping, which either underutilizes edge resources or causes high-latency cloud fallback for long outputs. This paper presents TAILOR, a token-aware inference framework that jointly optimizes offline partitioning and online routing for autoregressive transformer serving. TAILOR builds token-length buckets, generates memory-feasible partitioning strategies along the latency–memory trade-off frontier, selects a compact subset under edge capacity constraints, and routes each request according to predicted output length and device availability. Experiments with GPT-2 models on a heterogeneous edge–cloud testbed show that TAILOR reduces end-to-end latency by up to 22.2%, improves throughput by up to 28.5%, and lowers OOM-induced fallback by up to 73.1% compared with static partitioning baselines, demonstrating robust inference under long-tailed token workloads.**

I. Introduction

Transformer-based LLMs have become the foundation of interactive applications such as dialogue generation, code completion, and question answering [1]–[3]. Their inference cost, however, makes direct deployment on resource-constrained edge devices difficult [4]. Collaborative edge–cloud (CEC) inference addresses this challenge by partitioning model layers across edge devices and cloud servers, allowing latency-sensitive computation to remain close to users while the cloud handles overflow or expensive requests [5]–[7].

Decoder-only transformer inference is autoregressive: tokens are generated sequentially and previous key-value (KV) states are cached and reused [8], [9]. As a result, the cost of a request depends not only on model size and input length, but also on the output length, which is unknown when the request arrives. Longer outputs increase decoding steps, enlarge KV-cache footprints, and amplify tail latency. On memory-constrained edge devices, feasibility is determined by the peak KV memory realized during decoding; an underestimated output length can therefore cause OOM and force the request to fall back to the cloud.

979-8-3195-1703-6/26/$31.00 ©2026 IEEE

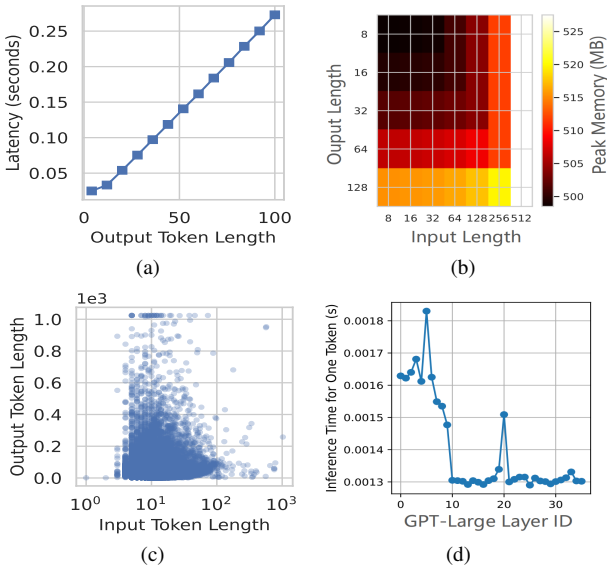


Fig. 1: Motivating observations: (a) latency increases with output length; (b) KV memory scales with input and output lengths; (c) output length distribution is long-tailed; (d) per-layer inference latency on the same device varies significantly.

Fig. 1 illustrates these effects using Databricks-Dolly-15k [10] and GPT-2 profiling results. Latency increases with generated output length, KV memory consumption grows with both input and output lengths, output lengths follow a pronounced long-tail distribution, and per-layer latency differs substantially even on the same edge device. These observations suggest that a single static partition is fundamentally mismatched with dynamic token-level workloads. Worst-case partitions must reserve excessive memory for rare long generations, leading to poor edge utilization, whereas average-case partitions may trigger OOM-induced fallback under long-output requests. Existing CEC systems such as EdgeShard and Jupiter [11], [12] still rely on static model-to-device mappings, and thus cannot explicitly exploit token-level heterogeneity across requests, layers, and runtime memory demands.

We propose *TAILOR*, a token-aware framework for transformer inference in CEC systems. TAILOR combines offline deployment planning with online request routing. Offline, it profiles token-length distributions, layer latency, KV-cache memory, and link bandwidth; builds a bucket-level latency–memory model; and selects a compact set of memory-feasible strategies under edge capacity constraints. Online, it predicts

each request's output length, maps the request to a token bucket, and dispatches it to a compatible strategy while accounting for device conflicts and latency targets. The contributions are threefold:

- We identify the limitations of static partitioning under dynamic token-length workloads and propose TAILOR, a token-aware inference framework for collaborative edge–cloud environments. TAILOR incorporates token variability into model partitioning and runtime scheduling.
- TAILOR introduces a token-bucket abstraction to capture input and output lengths, and uses Pareto-driven optimization to generate partitioning strategies that expose latency–memory tradeoffs under workload distributions.
- We design a token-aware partitioning and allocation framework that generates Pareto-diverse layer-to-device strategies and selects a memory-feasible subset for deployment. We further integrate a latency-aware online scheduler to dispatch requests safely and efficiently.
- We implement TAILOR on real LLMs and deploy it on a collaborative edge–cloud testbed. Experiments show up to 20% latency reduction and 4× fewer OOM events compared with static baselines such as EdgeShard.

II. System Model and Problem

A. Token-Aware Workload Model

We consider a decoder-only transformer with L sequential layers, indexed by $\ell \in \{1, \dots, L\}$. Each request has input length T_{in} and output length T_{out}. Since T_{out} is unknown at arrival and varies widely across requests, TAILOR groups requests into K token-length buckets $\mathcal{B} = \{B_1, \dots, B_K\}$. Bucket B_k has empirical probability p_k and representative provisioned length $\bar{T}_k$, obtained from historical traces. This abstraction preserves token-level heterogeneity while keeping deployment optimization tractable.

B. Edge–Cloud Execution Model

The CEC environment contains edge devices and a cloud server. Each device d has memory capacity M_d, profiled per-layer latency $t^{\ell}_{\text{comp}}(d)$, and measured bandwidth $B_{d,d'}$ to other devices. A strategy x assigns each layer to a device, where $x_{\ell d} = 1$ if layer ℓ runs on device d. For bucket B_k, the latency of strategy x is

$$T^{(k)}_{\text{lat}} = \sum_{\ell=1}^{L} \sum_{d \in \mathcal{D}} x_{\ell d} \cdot t^{\ell}_{\text{comp}}(d) \cdot \bar{T}_k + \sum_{\ell=2}^{L} \sum_{d,d' \in \mathcal{D}} x_{\ell-1,d} \cdot x_{\ell d'} \cdot t^{\ell-1,d\to d'}_{\text{comm}}(\bar{T}_k), \quad (1)$$

where $t^{\ell-1,d\to d'}_{\text{comm}}(\bar{T}_k)$ captures activation transmission latency and is zero if consecutive layers remain on the same device.

Memory feasibility is dominated by model parameters and KV cache. Let m_ℓ be the parameter memory of layer ℓ, and c_ℓ be the per-token KV-cache size. The provisioned memory on device d for bucket B_k is

$$\text{Mem}^{(k)}_d(x) = \sum_{\ell=1}^{L} x_{\ell d} \left(m_\ell + c_\ell \bar{T}_k \right). \quad (2)$$

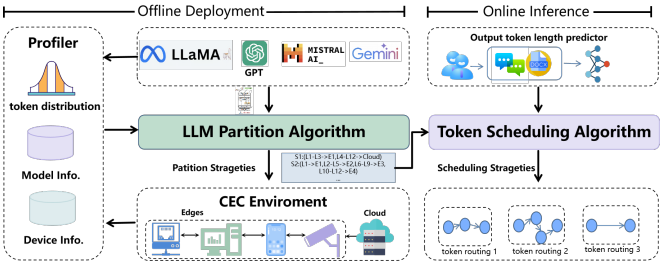


Fig. 2: The TAILOR framework. Offline planning profiles token distributions and system costs, generates and selects memory-feasible strategies, and deploys them on edge devices. Online routing predicts output length and dispatches each request to a compatible strategy or cloud fallback.

Using a safety factor $\beta_d \in (0, 1)$, a strategy is memory-feasible for bucket B_k only if $\text{Mem}^{(k)}_d(x) \le \beta_d M_d$ on every edge device it uses. Otherwise, the request may trigger OOM and fall back to the cloud. The design objective is therefore to minimize latency while maximizing token-bucket coverage by memory-feasible strategies.

III. TAILOR Design

Fig. 2 shows the two-stage design of TAILOR. The offline stage constructs a small strategy pool that covers common token regimes under memory constraints. The online stage uses this pool to route requests without expensive per-request model reconfiguration.

Design rationale. TAILOR decouples slow-timescale deployment from fast-timescale request dispatch. Recomputing layer placement for every request is impractical on resource-constrained edge devices, as it would require weight movement, buffer reallocation, and possible pipeline reconfiguration. In contrast, committing to a single static placement ignores the skewed nature of token workloads: most requests are short, while a small fraction of long requests dominate KV-cache pressure and tail latency. TAILOR therefore exposes a compact set of pre-deployed strategies to the online scheduler, enabling request-granularity adaptation while keeping the physical deployment stable.

TAILOR further optimizes for bucket coverage rather than uniform strategy utilization. Under long-tailed workloads, one strategy may cover frequent short or medium requests, whereas a few additional strategies are needed to handle rare long-output cases or transient device conflicts. TAILOR therefore allocates scarce edge memory according to frequency-weighted bucket coverage, instead of forcing all strategies to carry comparable traffic.

A. Offline Strategy Generation

TAILOR first generates candidate strategies that span the latency–memory trade-off frontier. It initializes a seed pool by greedily packing contiguous transformer layers onto devices under the largest provisioned token length $\bar{T}_{\max} = \max_k \bar{T}_k$, producing conservative memory-safe mappings. Starting from these seeds, a Pareto-based refinement searches over layer-to-device assignments. Each candidate is evaluated by expected

Algorithm 1: Token-Aware Strategy Generation

Input: Layers $\mathcal{L}$, devices $\mathcal{D}$, buckets $\mathcal{B}$, lengths $\{\bar{T}_k\}$
Output: Pareto strategy pool $\mathcal{S}$ and admissibility matrix δ

Initialize seed pool by greedily packing contiguous layers under $\bar{T}_{\max} = \max_k \bar{T}_k$;
for *each generation* **do**
  Evaluate every candidate by expected latency and overflow risk over all buckets;
  Apply non-dominated sorting and crowding-distance ranking;
  Generate offspring through structure-preserving crossover and mutation;
  Repair fragmented mappings by merging layer segments into adjacent devices;
  Keep elite candidates to form the next population;
foreach *strategy* j *and bucket* B_k **do**
  Set $\delta_{jk} = 1$ if j is zero-overflow for B_k and within the latency slack factor; otherwise set $\delta_{jk} = 0$;
return $\mathcal{S}$, δ;

Algorithm 2: Capacity-Constrained Strategy Selection

Input: Strategy pool $\mathcal{S}$, bucket probabilities p_k, admissibility matrix δ, device capacities
Output: Selected deployment set S

Initialize selected set $S \leftarrow \emptyset$ and uncovered bucket set $U \leftarrow \mathcal{B}$;
while U *is non-empty* **do**
  For every unselected strategy, compute its marginal coverage gain $\sum_{k \in U} p_k \delta_{jk}$;
  Estimate incremental memory after sharing already resident layers and updating peak KV reservation;
  Discard strategies that violate any edge-device capacity constraint;
  Select the feasible strategy with the largest gain-per-memory ratio;
  if *no feasible strategy exists* **then**
    break
  Add it to S and update covered buckets and device memory states;
return S;

latency and overflow risk across token buckets. Candidates that violate layer contiguity are repaired by merging fragmented segments into adjacent devices. The output is a Pareto-approximate strategy set $\mathcal{S}$.

For each strategy $j \in \mathcal{S}$ and bucket B_k, TAILOR computes a binary admissibility indicator δ_{jk}. We set $\delta_{jk} = 1$ only if the strategy is strictly memory-feasible for B_k and its latency is within a small slack factor of the best feasible latency for that bucket. Thus, the online scheduler only sees strategies that are both safe and efficient for the corresponding token regime. This strict admissibility test is important because OOM is not a benign local event in CEC inference: once an edge execution fails, the request must be restarted or recovered on the cloud, paying both the failed edge attempt and the cloud access delay. By filtering unsafe mappings offline, TAILOR converts a difficult runtime memory-risk problem into a deterministic lookup over safe bucket–strategy pairs.

The generation stage is intentionally performed offline. It can therefore use empirical profiling rather than coarse analytical FLOP models, which is important for edge devices whose latency is affected by memory bandwidth, kernel implementation, and communication overheads. Its output is not a single partition, but a small pool of alternatives that cover different token regimes. This design avoids online weight migration: model weights and KV-cache buffers are provisioned before serving starts, while online decisions only select among already deployed strategies.

B. Capacity-Constrained Strategy Selection

Only a limited number of strategies can be deployed because edge memory must store model weights and reserve KV-cache capacity. TAILOR selects a subset of strategies to maximize frequency-weighted bucket coverage:

$$\max \sum_{k=1}^{K} p_k z_k, \tag{3}$$

where $z_k = 1$ if bucket B_k is covered by at least one deployed strategy with $\delta_{jk} = 1$. The selection respects per-device memory constraints, with static parameters shared across strategies and KV cache provisioned for the largest bucket served by each deployed strategy. We use a greedy approximation that iteratively chooses the strategy with the largest marginal coverage gain per incremental memory cost, while explicitly checking device-level capacity. This deployment rule prioritizes frequent token regimes and keeps rare long-output regimes covered when memory permits.

This greedy rule follows the structure of monotone submodular coverage: adding a strategy can only increase the set of covered buckets, and its marginal gain decreases as more buckets are already covered. Under an abstract single-budget knapsack model, the standard gain-per-cost greedy rule gives a constant-factor approximation to the optimal frequency-weighted coverage. In the implemented multi-device setting, TAILOR keeps the same coverage principle but performs explicit per-device feasibility checks, which is more important than a closed-form guarantee for safe deployment on memory-constrained edge nodes.

C. Online Routing and Scheduling

At runtime, a lightweight predictor estimates the output length $\hat{T}_{\text{out}}$ and maps the request to bucket B_k. For every deployed admissible strategy j, the scheduler estimates the earliest completion time

$$C_j = \max_{d \in \text{Dev}(x^{(j)})} \text{BusyUntil}_d + T_{\text{lat}}^{(k,j)}, \tag{4}$$

where $\text{Dev}(x^{(j)})$ is the set of edge devices used by strategy j. The request is assigned to the admissible strategy with the smallest C_j that satisfies the bucket latency target. If no such strategy exists, the request is executed by the cloud-hosted model. This policy accounts for device conflicts, avoids unsafe edge execution, and preserves availability under contention or

TABLE I: CEC testbed configuration.

Role	Device	Memory	Count
Cloud	RTX 4060 Ti	8 GB	1
Edge	Jetson Orin Nano	4 / 8 GB	5 / 3

TABLE II: GPT-2 models used in evaluation.

Model	Layers	Params	File size
GPT2-Small	12	137M	0.5 GB
GPT2-Medium	24	320M	1.5 GB
GPT2-Large	36	812M	3.2 GB
GPT2-XL	48	1.6B	6.4 GB

prediction errors. Because all deployed strategies have already passed the bucket-level memory check, the online scheduler does not solve a placement problem or recompute KV feasibility. It only performs a bounded scan over admissible strategies, which keeps request dispatch lightweight enough for interactive serving.

TAILOR handles prediction uncertainty through conservative bucket provisioning and cloud fallback. The representative length $\bar{T}_k$ can be chosen as a high percentile within each bucket, so small prediction errors do not immediately violate memory constraints. If no safe or timely edge strategy is available, the scheduler proactively falls back to the cloud before starting unsafe edge execution.

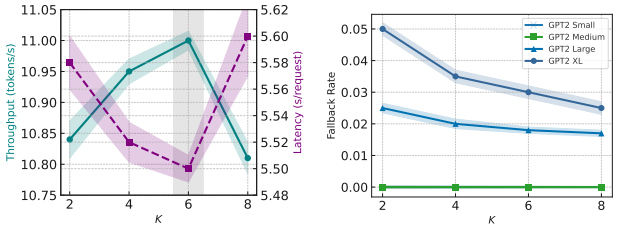
IV. Implementation and Evaluation

A. Experimental Setup

We implement TAILOR on a heterogeneous CEC testbed with eight NVIDIA Jetson Orin Nano boards and one RTX 4060 Ti GPU server as the cloud. Devices are connected through a gigabit Ethernet network, and Linux TC [13] is used to emulate bandwidth and propagation delays. Table I summarizes the hardware configuration. The edge side contains both 4GB and 8GB Jetson boards, which creates heterogeneous memory constraints and makes static worst-case partitioning inefficient. We evaluate GPT2-Small, GPT2-Medium, GPT2-Large, and GPT2-XL [14] on Databricks-Dolly-15k [15]; the model sizes are listed in Table II. We compare TAILOR with three baselines: Uniform, which evenly partitions layers across edge devices; EdgeShard [11], a representative static partitioning framework; and TAILOR-A, which uses strategy generation but removes capacity-constrained allocation optimization. We report average end-to-end latency, token throughput, and OOM-induced fallback rate. The fallback metric counts requests ultimately served by the cloud because the selected or available edge execution path is memory-infeasible under KV-cache provisioning. It directly captures whether a method can keep long-tailed token workloads on the edge without sacrificing safety.

B. Bucket Granularity and Prediction

Fig. 3 evaluates the impact of bucket granularity. Increasing K from 2 to 6 improves latency and throughput because requests are better matched to token-aware strategies. When



(a) Latency and throughput (b) Fallback rate

Fig. 3: Impact of token-bucket granularity.

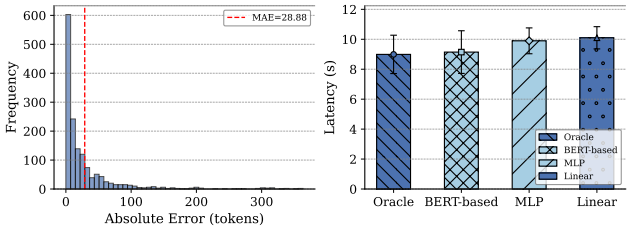


Fig. 4: Evaluation of the token-length predictor. The BERT-based predictor has small bucket-level error and yields latency close to an oracle using true output lengths.

$K = 8$, the benefits taper off and fallback increases for larger models because more fine-grained buckets require more strategies, while memory limits restrict deployment coverage. We use $K = 6$ as the default setting. TAILOR uses a BERT-based regressor to predict output length. Its mean latency is only 1.7% slower than an oracle that uses ground-truth output length, indicating that the bucket-level design is robust to practical prediction error.

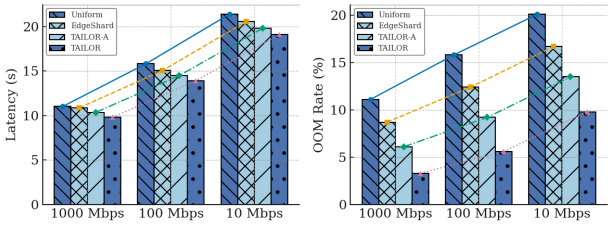
Fig. 4 further examines prediction quality. The predictor is trained on instruction text using a BERT encoder followed by an MLP regressor, with a logarithmic transform applied to the output length to reduce the effect of heavy-tailed labels. The mean absolute error is 28.88 tokens. More importantly, the scheduling impact is small: the BERT-based predictor produces nearly the same latency as the oracle and clearly outperforms simpler MLP and linear predictors. This confirms that TAILOR does not require exact token-length estimation; it only needs sufficiently accurate bucket assignment to choose a compatible deployment strategy.

C. End-to-End Performance

Table III reports the main end-to-end results. TAILOR reduces latency by 22.2% on GPT2-Small and 5.3% on GPT2-XL compared with Uniform, and consistently outperforms EdgeShard. The gain is larger on smaller models because edge memory can host multiple specialized strategies, while larger models leave less deployment flexibility. TAILOR also substantially reduces OOM-induced fallback. Relative to Uniform, fallback decreases from 9.3% to 2.5% on GPT2-Small and from 13.5% to 4.1% on GPT2-XL. These results show that token-aware deployment improves both latency and robustness. Throughput improves as well because TAILOR keeps suitable strategies resident on edge devices and reduces high-latency cloud fallback.

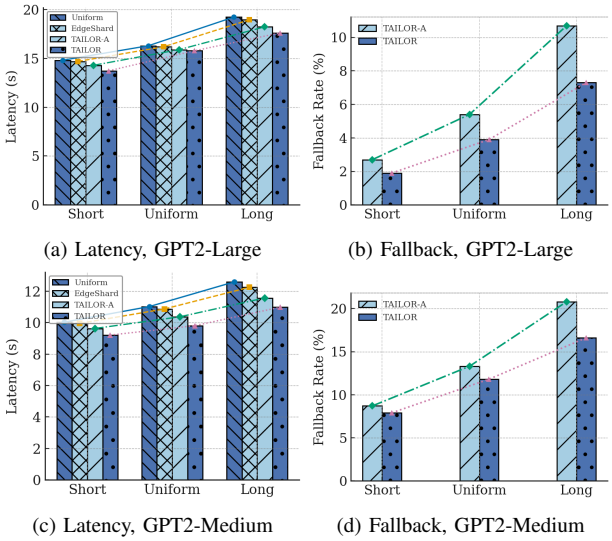
TABLE III: End-to-end performance comparison across GPT-2 model sizes. The percentage in parentheses is the relative change with respect to Uniform. OOM-induced fallback rate is the fraction of requests offloaded to the cloud due to edge-side memory infeasibility.

Algorithms	GPT2-Small			GPT2-Medium		
	Throughput (token/s)	Latency (s)	OOM-induced fallback rate	Throughput (token/s)	Latency (s)	OOM-induced fallback rate
Uniform	10.25 (+0.00%)	5.90 (-0.00%)	9.3% (-0.00%)	5.49 (+0.00%)	11.02 (-0.00%)	11.1% (-0.00%)
EdgeShard	10.92 (+6.53%)	5.54 (-6.10%)	7.8% (-16.1%)	5.56 (+1.27%)	10.87 (-1.36%)	8.7% (-21.6%)
TAILOR-A	11.14 (+8.68%)	5.43 (-7.96%)	4.8% (-48.4%)	5.83 (+6.19%)	10.37 (-5.89%)	6.1% (-45.0%)
TAILOR	**13.18 (+28.5%)**	**4.59 (-22.2%)**	**2.5% (-73.1%)**	**6.16 (+12.2%)**	**9.81 (-10.9%)**	**3.3% (-70.3%)**
Algorithms	GPT2-Large			GPT2-XL		
	Throughput (token/s)	Latency (s)	OOM-induced fallback rate	Throughput (token/s)	Latency (s)	OOM-induced fallback rate
Uniform	3.71 (+0.00%)	16.29 (-0.00%)	12.3% (-0.00%)	2.84 (+0.00%)	21.28 (-0.00%)	13.5% (-0.00%)
EdgeShard	3.73 (+0.53%)	16.19 (-0.61%)	9.6% (-22.0%)	2.88 (+1.40%)	20.96 (-1.50%)	10.2% (-24.4%)
TAILOR-A	3.80 (+2.42%)	15.89 (-2.45%)	6.9% (-43.9%)	2.96 (+4.22%)	20.38 (-4.22%)	7.5% (-44.4%)
TAILOR	**3.83 (+3.23%)**	**15.78 (-3.13%)**	**3.8% (-69.1%)**	**3.00 (+5.63%)**	**20.16 (-5.26%)**	**4.1% (-69.6%)**



(a) Latency (b) Fallback rate

Fig. 5: Impact of edge-to-cloud bandwidth on GPT2-Medium.



(a) Latency, GPT2-Large (b) Fallback, GPT2-Large

(c) Latency, GPT2-Medium (d) Fallback, GPT2-Medium

Fig. 6: Sensitivity to token-length distributions. TAILOR adapts to short-biased, uniform, and long-biased workloads.

The comparison with TAILOR-A isolates the value of allocation optimization. TAILOR-A already benefits from having multiple candidate strategies, but its deployment is not guided by bucket probability or shared-memory constraints. As a result, it may deploy strategies that cover overlapping frequent buckets while leaving costly long-output buckets uncovered. TAILOR avoids this failure mode by selecting strategies according to marginal frequency-weighted coverage under per-device memory limits. The improvement is therefore not only due to generating more partitions; it comes from deploying the right subset of partitions for the observed token distribution.

D. Robustness under Workload and Network Variation

TAILOR remains effective under changing token-length distributions. On short-biased, uniform, and long-biased workloads derived from Databricks-Dolly-15k, TAILOR consistently achieves the lowest latency and maintains lower fallback rates than Uniform, EdgeShard, and TAILOR-A. The advantage persists even for long-biased workloads, where KV-cache pressure is highest, because TAILOR routes long-output requests to compatible strategies or offloads them early before severe queueing delay or OOM occurs.

Fig. 6 reports the detailed results on GPT2-Large and GPT2-Medium. Under short-biased workloads, TAILOR benefits from routing common short requests to low-latency strategies rather than reserving all edge memory for rare long outputs. Under uniform and long-biased workloads, its allocation stage becomes more important: it keeps enough coverage for longer buckets and avoids selecting strategies that look fast on average but are unsafe when KV cache grows. TAILOR-A improves over static baselines because it has multiple strategies, but its random deployment choice leaves some important buckets uncovered. The gap between TAILOR and TAILOR-A therefore demonstrates that capacity-constrained strategy selection is necessary, not merely the existence of multiple candidate partitions.

Fig. 5 evaluates performance under constrained edge-to-cloud bandwidth. As bandwidth decreases from 1,000 Mb/s to 10 Mb/s, all methods experience higher latency and fallback due to increasingly expensive cloud access. TAILOR remains the most robust across all bandwidth settings: at 10 Mb/s, it reduces latency by 19.1%, 12.4%, and 4.5% compared with Uniform, EdgeShard, and TAILOR-A, respectively, and achieves lower fallback. This result confirms that token-aware strategy selection is useful when cloud fallback is costly.

Practical considerations. TAILOR separates slow deployment decisions from fast request scheduling. Profiling, bucket

construction, strategy generation, and weight loading are performed offline, while the online scheduler only scans the deployed admissible strategies and updates device busy timestamps. This avoids request-time weight migration, which would be costly on edge devices and could introduce memory fragmentation. The strategy set can be refreshed periodically when the workload distribution, model version, or available devices change. TAILOR is also complementary to quantization, batching, and KV-cache compression: these techniques reduce the memory footprint, whereas TAILOR decides which feasible partitions should be resident and how token-variable requests should be routed among them. For short-paper scope, we deliberately keep the runtime policy simple and deterministic; a longer version can integrate batching-aware admission control, adaptive bucket reconstruction, and tighter robustness analysis under prediction errors.

V. Related Work

Collaborative edge–cloud inference. Collaborative inference jointly uses edge and cloud resources to serve latency-sensitive AI workloads. Early systems such as Neurosurgeon [7], DADS [16], and Auto-Split [6] optimize split points or offloading decisions under heterogeneous computation and network conditions. Recent transformer-oriented systems, including EdgeShard [11] and Jupiter [12], further show that layer partitioning can improve generative LLM inference on edge devices, and adaptive LLM systems such as Splitwise [17] and DAPO [18] explore phase splitting or mobility-aware offloading. However, these approaches generally derive one static partition or adapt to system state without explicitly optimizing deployment around output-token buckets. They therefore cannot directly capture the long-tailed output-token variability and KV-cache growth that determine memory feasibility during autoregressive decoding.

Efficient LLM serving and memory-aware optimization. LLM serving systems improve throughput and latency through batching, KV-cache management, and phase separation, while compression techniques such as quantization [19] reduce model footprint. These techniques are complementary to TAILOR. Rather than changing the model architecture or optimizing a centralized serving cluster, TAILOR focuses on token-aware deployment in memory-constrained edge–cloud environments. It models output-length variability through token buckets, constructs multiple memory-feasible layer-to-device strategies, and routes each request to a compatible strategy according to predicted token length and device availability. This design directly targets OOM-induced fallback and tail-latency instability caused by dynamic token workloads.

VI. Conclusion

This paper presented TAILOR, a token-aware partitioning and routing framework for edge–cloud transformer inference. TAILOR uses token-length buckets to model output variability, constructs memory-feasible layer-to-device strategies offline, selects a compact deployment set under edge capacity constraints, and schedules online requests according to predicted token length and device availability. Experiments on a heterogeneous edge–cloud testbed show that TAILOR reduces latency, improves throughput, and substantially lowers OOM-induced fallback compared with static partitioning baselines. These results demonstrate that token variability should be treated as a first-class factor in the design of collaborative edge–cloud LLM serving systems.

References

[1] F. F. Xu, U. Alon, G. Neubig, and V. J. Hellendoorn, "A systematic evaluation of large language models of code," in *Proc. ACM SIGPLAN Int. Symp. Mach. Program.*, 2022, pp. 1–10.

[2] R. Qian, X. Dong, P. Zhang, Y. Zang, S. Ding, D. Lin, and J. Wang, "Streaming long video understanding with large language models," in *Adv. Neural Inf. Process. Syst.*, vol. 37, 2024, pp. 119336–119360.

[3] A. Borzunov, M. Ryabinin, A. Chumachenko, D. Baranchuk, T. Dettmers, Y. Belkada, P. Samygin, and C. A. Raffel, "Distributed inference and fine-tuning of large language models over the internet," in *Adv. Neural Inf. Process. Syst.*, vol. 36, 2023, pp. 12312–12331.

[4] D. Narayanan, J. Hegeman, A. Phanishayee, and M. Zaharia, "Efficient large-scale language model training on GPU clusters using Megatron-LM," in *Proc. MLSys Conf.*, 2021.

[5] C. Hu and B. Li, "When the edge meets transformers: Distributed inference with transformer models," in *Proc. IEEE Int. Conf. Distrib. Comput. Syst. (ICDCS)*, 2024, pp. 82–92.

[6] A. Banitalebi-Dehkordi, N. Vedula, J. Pei, F. Xia, L. Wang, and Y. Zhang, "Auto-split: A general framework of collaborative edge-cloud AI," in *Proc. ACM SIGKDD Conf. Knowl. Discov. Data Min. (KDD)*, 2021, pp. 2543–2553.

[7] Y. Kang, J. Hauswald, C. Gao, A. Rovinski, T. Mudge, J. Mars, and L. Tang, "Neurosurgeon: Collaborative intelligence between the cloud and mobile edge," *ACM SIGARCH Comput. Archit. News*, vol. 45, no. 1, pp. 615–629, 2017.

[8] A. Santilli, S. Severino, E. Postolache, V. Maiorca, M. Mancusi, R. Marin, and E. Rodola, "Accelerating transformer inference for translation via parallel decoding," arXiv:2305.10427, 2023.

[9] Y. Liu, H. Li, Y. Cheng, S. Ray, Y. Huang, Q. Zhang, K. Du, J. Yao, S. Lu, G. Ananthanarayanan, and J. Jiang, "CacheGen: KV cache compression and streaming for fast large language model serving," in *Proc. ACM SIGCOMM Conf.*, 2024, pp. 38–56.

[10] Databricks, "Databricks Dolly 15k dataset," 2023. [Online]. Available: https://huggingface.co/datasets/databricks/databricks-dolly-15k

[11] M. Zhang, X. Shen, J. Cao, Z. Cui, and S. Jiang, "EdgeShard: Efficient LLM inference via collaborative edge computing," *IEEE Internet Things J.*, 2024.

[12] S. Ye, B. Ouyang, L. Zeng, T. Qian, X. Chu, J. Tang, and X. Chen, "Jupiter: Fast and resource-efficient collaborative inference of generative LLMs on edge devices," in *Proc. IEEE INFOCOM*, 2025, pp. 1–10.

[13] B. Hubert *et al.*, "Linux advanced routing & traffic control HOWTO," Netherlabs BV, 2002.

[14] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, and I. Sutskever, "Language models are unsupervised multitask learners," OpenAI Tech. Rep., 2019.

[15] M. Conover, M. Hayes, A. Mathur, J. Xie, J. Wan, S. Shah, A. Ghodsi, P. Wendell, M. Zaharia, and R. Xin, "Free Dolly: Introducing the world's first truly open instruction-tuned LLM," Databricks Blog, 2023. [Online]. Available: https://www.databricks.com/blog/2023/04/12/dolly-first-open-commercially-viable-instruction-tuned-llm

[16] C. Hu, W. Bao, D. Wang, and F. Liu, "Dynamic adaptive DNN surgery for inference acceleration on the edge," in *Proc. IEEE INFOCOM*, 2019, pp. 1423–1431.

[17] P. Patel, E. Choukse, C. Zhang, A. Shah, I. Goiri, S. Maleki, and R. Bianchini, "Splitwise: Efficient generative LLM inference using phase splitting," in *Proc. ACM/IEEE Int. Symp. Comput. Archit. (ISCA)*, 2024.

[18] H. Feng, G. Huang, N. Zhou, F. Zhang, Y. Liu, X. Zhou, and J. Liu, "DAPO: Mobility-aware joint optimization of model partitioning and task offloading for edge LLM inference," *Electronics*, vol. 14, no. 19, p. 3929, 2025.

[19] B. Jacob, S. Kligys, B. Chen, M. Zhu, M. Tang, A. Howard, H. Adam, and D. Kalenichenko, "Quantization and training of neural networks for efficient integer-arithmetic-only inference," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, 2018, pp. 2704–2713.