IWQoS 2026

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

huangxy32@chinatelecom.cn · remington.liu@outlook.com · jiewu@temple.edu

Background: LLM Inference in Edge–Cloud Systems

Collaborative Edge–Cloud (CEC) Inference

Partition model layers across edge devices and cloud servers, allowing latency-sensitive computation to remain close to users while the cloud handles overflow.

Key systems: Neurosurgeon, DADS, Auto-Split, EdgeShard, Jupiter.

Autoregressive Transformer Inference

Tokens generated sequentially; previous key-value (KV) states are cached and reused.

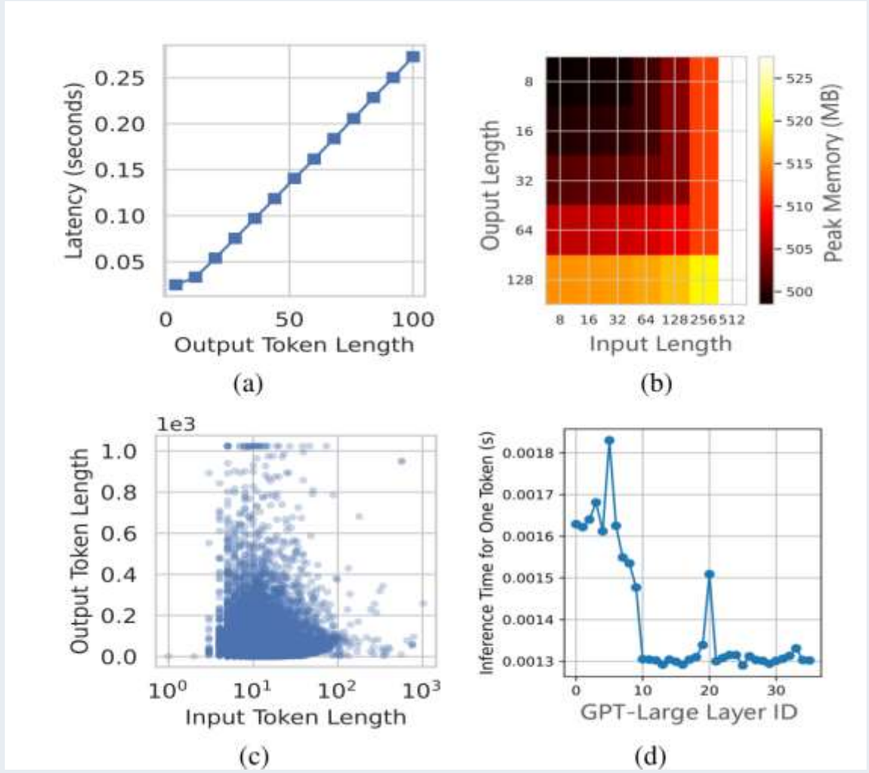
Cost depends on model size, input length, and **output length**. Output length is unknown at request arrival and varies widely.

Challenge: Token-Level Variability

1. Output lengths are **unknown** at arrival, making memory planning impossible
2. KV-cache memory **grows linearly** during decoding with both input and output lengths
3. Long-tailed requests cause **OOM failures** on memory-constrained edge devices

Core Insight: A single static partition is fundamentally mismatched with dynamic token-level workloads. Worst-case partitions waste edge memory; average-case partitions risk OOM under long outputs.

Motivating Observations: Token-Level Variability



Profiling results on Databricks-Dolly-15k with GPT-2

(a) Latency increases with output length

More decoding steps directly increase end-to-end latency.

(b) KV memory scales with input/output lengths

Peak KV-cache grows with both input and output token lengths.

(c) Output length is long-tail distributed

Most requests are short; rare long requests dominate KV-cache.

(d) Per-layer latency varies significantly

Inference latency differs even on the same device.

Core Insight: These observations demonstrate that a single static partition is fundamentally mismatched with dynamic token-level workloads. Existing CEC systems rely on static mappings and cannot exploit token-level heterogeneity.

Problem: Limitations of Static Partitioning

Worst-Case Partitions

Reserve excessive memory for rare long generations:

- Poor edge resource utilization
- Underutilized memory for short requests
- High cost for limited gain

Average-Case Partitions

Optimize for typical request lengths:

- Trigger OOM under long-output requests
- Force expensive cloud fallback
- Failed edge attempts add latency penalty

Existing Systems Cannot Exploit Token-Level Heterogeneity

EdgeShard and Jupiter derive one static partition without optimizing around output-token buckets. Splitwise and DAPO explore phase splitting or mobility-aware offloading, but do not explicitly model token-length variability.

Our Approach: TAILOR

A token-aware inference framework that jointly optimizes **offline partitioning** and **online routing**. Builds token-length buckets, generates memory-feasible strategies, and routes requests by predicted output length.

System Model & Problem Formulation

A. Token-Aware Workload Model

Decoder-only transformer with L layers. Each request has input T_{in} and **unknown** output T_{out}. Grouped into K buckets B with probability p_k and length $\bar{T}_k$.

B. Edge–Cloud Execution Model

Device d has memory M_d, latency $t_{comp}^{\ell}(d)$, bandwidth $B_{d,d'}$. Strategy x assigns layer ℓ to device d via $x_{\ell d} \in \{0,1\}$.

Communication cost depends on consecutive layers on different devices, weighted by token length $\bar{T}_k$.

Latency model for bucket B_k

$$T_{lat}^{(k)}(x) = \underbrace{\sum_{\ell,d} x_{\ell d}\, t_{comp}^{\ell}(d)\, \bar{T}_k}_{compute} + \underbrace{\sum_{\ell,d,d'} x_{\ell-1,d}\, x_{\ell d'}\, t_{comm}^{\ell-1,d\to d'}(\bar{T}_k)}_{comm}$$

Memory feasibility

$$Mem_d^{(k)}(x) = \sum_{\ell} x_{\ell d}(m_\ell + c_\ell \bar{T}_k) \le \beta_d M_d$$

Optimization objective

$$max \sum_{k=1}^{K} p_k z_k \; s.t. \; z_k = 1[\exists j : \delta_{jk} = 1]$$

Find memory-feasible strategies maximizing bucket coverage, then route requests at runtime.

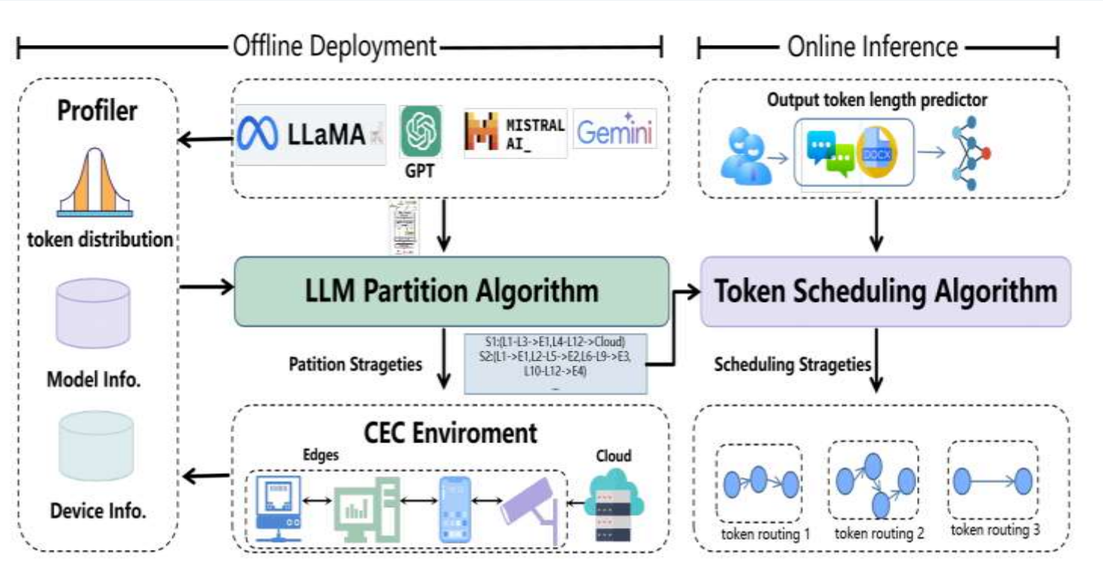
TAILOR Overview: Two-Stage Design

Offline: Deployment Planning

1. **Profile** token-length distributions, per-layer latency, KV-cache memory, link bandwidth
2. **Build** bucket-level latency–memory model
3. **Generate** memory-feasible Pareto strategies
4. **Select** compact set under edge capacity

Online: Request Routing

1. **Predict** output length via BERT regressor
2. **Map** request to token bucket
3. **Dispatch** to compatible strategy



The TAILOR framework architecture

Design Rationale: Decouples slow-timescale deployment from fast-timescale dispatch.

Offline: Strategy Generation & Selection

Algorithm 1: Strategy Generation

Input: Layers L, devices D, buckets B

Step 1 — Seed pool:

Greedily pack contiguous layers under $\bar{T}_{max}$.

Step 2 — Pareto refinement:

Non-dominated sorting + crowding-distance ranking. Crossover, mutation, repair.

Step 3 — Admissibility:

$\delta_{jk} = 1$ only if memory-feasible and within latency slack.

Output: Pareto pool S and δ

Algorithm 2: Capacity-Constrained Selection

Input: Pool S, probabilities p_k, admissibility δ

Step 1 — Greedy selection:

Select by largest marginal coverage gain per memory cost.

Step 2 — Feasibility check:

Estimate incremental memory with layer sharing. Discard if violates capacity.

Step 3 — Termination:

Stop when no feasible strategy or all buckets covered.

Output: Deployment set maximizing $\sum_k p_k z_k$

Key Design Insights

- **Strict admissibility δ_{jk}:** Converts runtime memory-risk into deterministic lookup over safe bucket–strategy pairs.
- **Monotone submodular coverage:** Greedy rule provides constant-factor approximation.
- **Offline execution:** No online weight migration; weights and KV-cache provisioned before serving.
- **Complexity**: Alg.1 O(ML+TPKL); Alg.2 O(SN(K+D)) per iteration.
- **Approximation (Thm.1):** Greedy selection achieves (1−1/e) ≈ 63% of optimal coverage.

Online Routing and Scheduling

1 Predict

BERT estimates $\hat{T}_{out}$

Maps to bucket B_k

→

2 Estimate

For each strategy j:

$C_j = max_d\, Busy_d + T_{lat}^{(k,j)}$

→

3 Select

Smallest C_j meeting latency target

Bounded scan over safe strategies

→

4 Execute

Edge execution

Or proactive cloud fallback

Handling Prediction Uncertainty

Conservative provisioning: $\bar{T}_k$ as high percentile; small errors do not violate memory.

Proactive fallback: Falls back to cloud before unsafe edge execution.

Key Properties

No weight migration: Provisioned before serving.

Deterministic safety: Bounded scan over pre-checked strategies.

Complementary: Works with quantization, batching, KV compression.

Design Principle: TAILOR separates slow-timescale deployment from fast-timescale dispatch. Strategy set refreshable when workload changes.

Experimental Setup

Hardware configuration

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

Models

Model	Layers	Params	Size
GPT2-S	12	137M	0.5 GB
GPT2-M	24	320M	1.5 GB
GPT2-L	36	812M	3.2 GB
GPT2-XL	48	1.6B	6.4 GB

Dataset: Databricks-Dolly-15k **Network:** Gigabit Ethernet with Linux TC

Buckets: K = 6 **Safety factor:** $\beta = 0.9$

Baselines:

Uniform: Evenly assigns layers across edge devices and performs standard pipeline scheduling.

EdgeShard: statically partitions the transformer using a dynamic programming algorithm on profiled latency and bandwidth, then performs pipeline scheduling.

TAILOR-A: TAILOR without Allocation Optimization (TAILOR-A).

Metrics

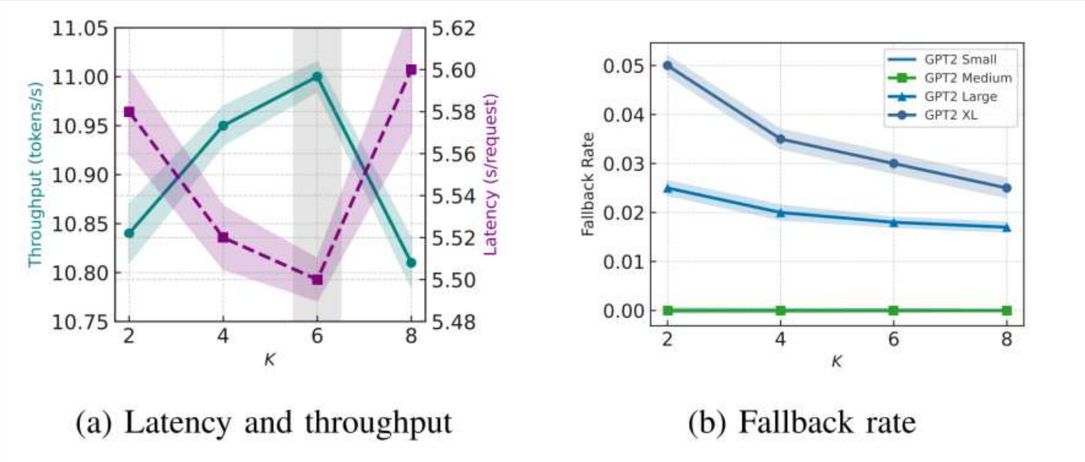
End-to-end latency
Time from request arrival to final token

Token throughput
Tokens processed per second

OOM fallback rate
Fraction offloaded due to edge infeasibility

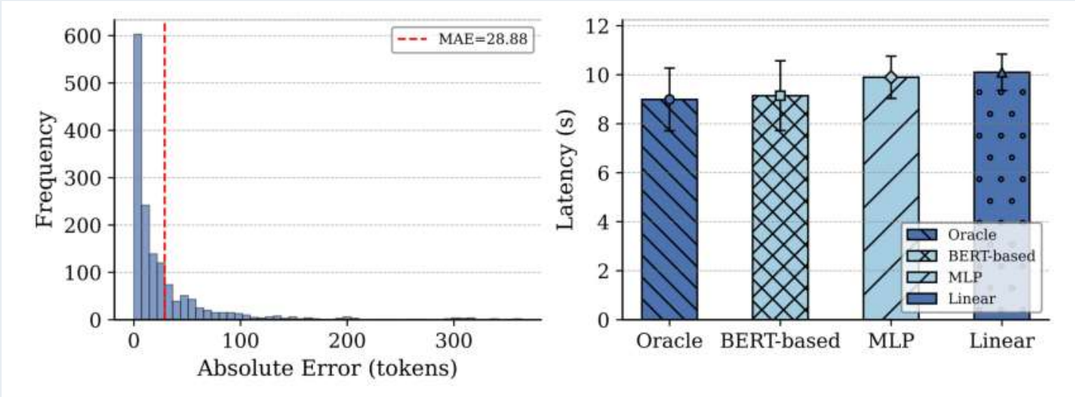
Bucket Granularity & Prediction

Impact of bucket granularity K



K = 2 to 6 improves latency and throughput. K = 8 tapers off due to memory limits. **K = 6 is default.**

Token-length predictor



BERT predictor: MAE = 28.88 tokens. Latency only **1.7% slower** than oracle. Bucket-level design is robust to prediction error.

Insight: Coarse buckets reduce provisioning overhead; fine buckets capture workload variance. K = 6 strikes the best balance.

Online Routing and Scheduling

1 Predict

BERT estimates $\hat{T}_{out}$

Maps to bucket B_k

→

2 Estimate

For each strategy j:

$C_j = max_d\, Busy_d + T_{lat}^{(k,j)}$

Compute max safe input/output lengths.

→

3 Select

Smallest C_j meeting latency target

Bounded scan over safe strategies

→

4 Execute

Edge execution

Or proactive cloud fallback

Handling Prediction Uncertainty

Conservative provisioning: $\bar{T}_k$ as high percentile; small errors do not violate memory.

Proactive fallback: Falls back to cloud before unsafe edge execution.

Key Properties

No weight migration: Provisioned before serving.

Deterministic safety: Bounded scan over pre-checked strategies.

Complementary: Works with quantization, batching, KV compression.

Design Principle: TAILOR separates slow-timescale deployment from fast-timescale dispatch. Strategy set refreshable when workload changes.

End-to-End Performance

Table III — Performance across GPT-2 models (change vs. Uniform)

Algorithm	GPT2-SLat. (s)	GPT2-SThr. (t/s)	GPT2-SFallback	GPT2-XLLat. (s)	GPT2-XLThr. (t/s)	GPT2-XLFallback	Verdict
Uniform	5.90	10.25	9.3%	21.28	2.84	13.5%	Baseline
EdgeShard	5.54	10.92	7.8%	20.96	2.88	10.2%	Modest
TAILOR-A	5.43	11.14	4.8%	20.38	2.96	7.5%	Good
TAILOR	**4.59**	**13.18**	**2.5%**	**20.16**	**3.00**	**4.1%**	**Best**

22.2%
Latency reduction

28.5%
Throughput gain

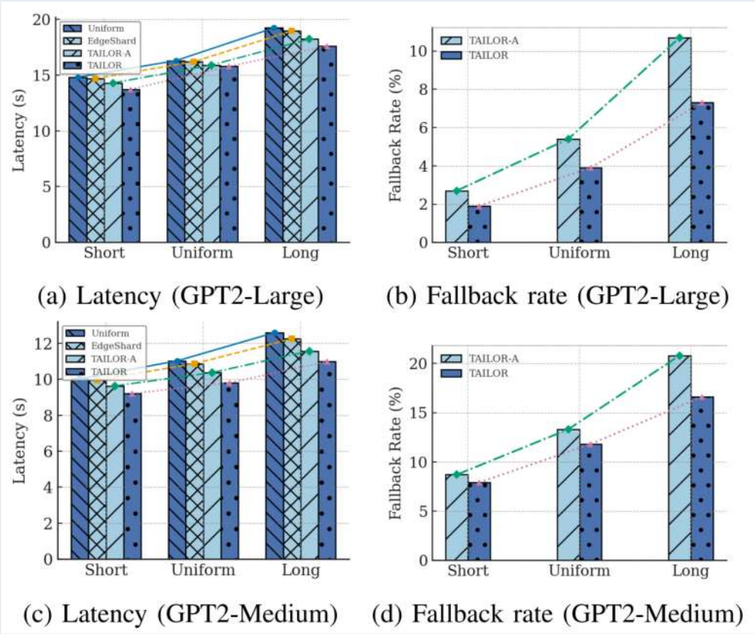
73.1%
Less OOM fallback

69.6%
Less OOM (GPT2-XL)

Gains larger on smaller models: edge memory hosts multiple specialized strategies. TAILOR consistently outperforms EdgeShard and TAILOR-A.

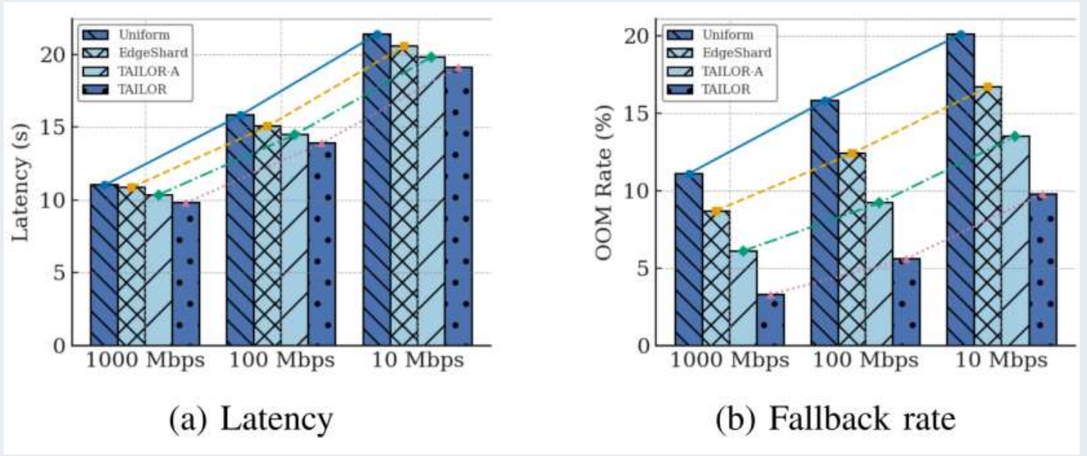
Robustness: Workload & Network Variation

Impact of edge-to-cloud bandwidth



At 10 Mb/s, TAILOR reduces latency by **19.1%** vs Uniform and **12.4%** vs EdgeShard. Most robust across all bandwidth settings.

Sensitivity to token-length distributions



Effective under short-biased, uniform, and long-biased workloads. Improvement comes from deploying the *right subset* of partitions, not merely generating more.

Takeaway: Capacity-constrained selection is essential. TAILOR-A generates strategies but leaves important buckets uncovered.

Conclusion

> TAILOR is a token-aware partitioning and routing framework for edge-cloud transformer inference, addressing the mismatch between static partitioning and dynamic token-level workloads.

Contributions

1. Identified limitations of static partitioning under token-length variability
2. Introduced token-bucket abstraction with Pareto-driven optimization
3. Designed capacity-constrained selection maximizing bucket coverage
4. Integrated latency-aware online scheduler with safety guarantees

Key Results

22.2% latency reduction

28.5% throughput gain

73.1% less OOM fallback

69.6% less OOM (GPT2-XL)

Future Work

Batching-aware admission control, adaptive bucket reconstruction, tighter robustness analysis under prediction errors, integration with quantization and KV-cache compression.

Thank You

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

IWQoS 2026

Q&A: huangxy32@chinatelecom.cn