Joint Partitioning and Placement of Foundation Models for Real-Time Edge AI

Aladin Djuhera*, Fernando Koch†, Alecio Binotto‡
*Technical University of Munich, Germany, †Florida Atlantic University, USA, ‡Carl Zeiss AG, Germany
Emails: aladin.djuhera@tum.de, kochf@fau.edu, alecio.binotto@zeiss.com

***Abstract*—Inference over large-scale foundation models within heterogeneous edge environments necessitates a fundamentally reconfigurable orchestration substrate. Static partitioning of model layers presumes temporal stability across compute and network resources, which is misaligned with the volatility of real-world deployments. We introduce a framework in which both the spatial placement and internal segmentation of foundation models are elevated to runtime-resolved constructs. The orchestration problem is formalized as a constrained optimization over layer-wise assignments, subject to evolving latency, utilization, and privacy gradients. The framework implements reactive inference composition responsive to infrastructural fluctuations by integrating model-aware capacity profiling with dynamic graph re-partitioning and reallocation. We introduce architectural and algorithmic components, along with a representative use case in 6G multi-access edge computing.**

***Index Terms*—Foundation Model Inference, Distributed Orchestration, Edge AI, 6G Networks**

I. Introduction

Next-generation 6G-enabled networks will need to support a multitude of AI services based on large foundation models (LFM), such as transformer-based large language models (LLM) [1]. However, deploying LFMs for inference requires significant compute, making their adoption challenging for edge environments [2]. *Distributed split inference* (DSI) [3] has emerged as a promising approach to alleviate the computational burden. It partitions an LFM into multiple segments that are executed sequentially across different nodes, e.g., some executed locally, while heavier segments can be outsourced. However, such *model splits* are mostly static and predetermined before execution, thus lacking adaptability to dynamic and heterogeneous operational conditions such as fluctuating network reliability or changing node utilization. Consequently, these approaches lead to suboptimal performance, compromising latency, resource utilization, and quality of service (QoS) guarantees, especially in mission-critical applications [4]. This problem becomes even more acute in *resource-constrained* and *heterogeneous* edge environments, where multiple users rely on accessing shared resources such as multi-access edge compute (MEC), and where data privacy regulations often restrict offloading computations to the cloud. Such volatility renders any *a priori* static split untenable.

This work was supported by the German Federal Ministry of Research, Technology and Space within the project 6G-life (Grant 16KISK002), by the Bavarian Ministry of Science and the Arts through the project Next Generation AI Computing (gAIn), and by the Bavarian Ministry of Economic Affairs, Regional Development and Energy through the project 6G Future Lab Bavaria.

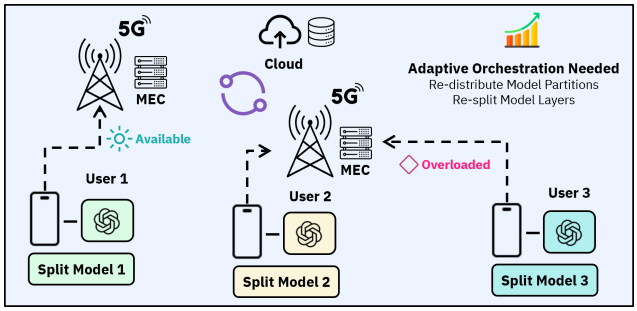

Fig. 1. DSI with LFMs requires adaptive orchestration between edge and cloud nodes to guarantee latency, service quality, and efficient node utilization.

In addition, existing inference orchestration frameworks, such as Kubernetes, Ray Serve, InferLine, and KubeEdge, excel at container or micro-batch scheduling but treat LFMs primarily as black boxes, thus lacking mechanisms for runtime layer re-partitioning or privacy-aware placement [5]. Consequently, the key problem of *joint model-aware partition and placement* under dynamic edge conditions remains open.

In this work, we address this problem by introducing an *adaptive split inference orchestration framework*, extending existing workload orchestrators with domain-specific capabilities specifically tailored for LFMs, such as (multi-modal) LLMs. We introduce the following capabilities by leveraging the modular architecture of these models:

1) **Distribution of workloads to edge nodes** with better performance or capacity than the original source node.
2) **Relocation of split LFM segments** to dynamically optimize resources under changing compute conditions.
3) **Dynamic LFM re-splitting** to further improve performance and resource utilization when required.

Unlike general-purpose workload schedulers, our framework operates on the *computational graph of the LFM itself*, allowing decisions at the granularity of, e.g., individual transformer blocks. This enables QoS-driven re-splitting that commodity orchestrators do not address. Furthermore, as our solution emphasizes split inference, *privacy* can be implemented as an additional feature at no cost if sensitive LFM layers can be executed locally, which makes reverse engineering data from model weights significantly more challenging for attackers [6]. Through this approach, we establish a foundation for *real-time* and *QoS-aware* LFM inference in edge networks, aligning with key 6G objectives of seamless connectivity, low inference latency, and intelligent edge resource management [7].

II. Background

A. Challenges in Edge AI Workload Orchestration

The current industry norm has been to integrate general-purpose orchestration platforms (e.g., Kubernetes or proprietary MEC orchestrators), which lack mechanisms to dynamically redistribute or reconfigure large model partitions based on real-time changes in *network conditions, node utilization*, or *connectivity* [8]. Further, while model-serving stacks such as Triton, InferLine, Ray Serve, and MLC-Serve introduce batching or replica autoscaling, they still treat neural networks as opaque binaries and cannot *re-partition* a model graph at runtime. Edge extensions (e.g., KubeEdge and OpenYurt) inherit the same pod-level abstraction, leaving the joint problem of *joint layer splitting and placement*, unaddressed. As a result:

- **Latency spikes** occur when critical links become congested, delaying real-time applications.
- **Straggler problems** arise when tasks are bottlenecked on overloaded or slower nodes, degrading overall QoS.
- **Resource utilization** becomes imbalanced, either overloading certain nodes or leaving others underutilized, leading to missed service-level agreements (SLAs).
- **Privacy risks** escalate when sensitive data must be offloaded remotely due to inadequate local processing.

While current research predominantly focuses on *efficient AI model training* [9], [10], the practical challenges of *efficient inference* remain relatively overlooked [11]. Yet, these challenges are increasingly critical for the widespread adoption of LFMs in industrial and commercial scenarios, particularly in future *AI as a Service* (AIaaS)-driven 6G networks [12].

B. Distributed and Adaptive Split Inference

DSI partitions a model across different compute locations (e.g., client device, MEC node, cloud), where often lightweight or privacy-sensitive model layers are executed on-device and subsequent layers are offloaded to a remote server [13].

Although DSI enables LFMs to operate closer to data sources, current implementations predominantly employ static splits defined *a priori* based on expected conditions without runtime adaptation. This becomes problematic when, for example, a single MEC-enabled base station becomes saturated by various other inference workloads [14] (see Fig. 1). While some studies, such as EdgeShard [15], explore collaborative inference setups where a model is shared across edge nodes, these approaches continue to lack dynamic orchestration of model splits and thus cannot effectively respond to real-world changes, such as fluctuating node workloads or intermittent connectivity, resulting in suboptimal latency, inefficient resource utilization, and degraded service quality [16]. Thus, in standard implementations, orchestrators cannot dynamically decide to offload additional LFM layers to another edge node or re-split the neural network, leaving significant performance and reliability gains unrealized. This is especially problematic for heterogeneous compute nodes. Thus, a *one-size-fits-all* static partitioning rarely works, as local workloads, performance constraints, and available resources tend to differ [2].

However, recent research highlights the benefits of *adaptive split inference* wherein partition points or even partition strategies (e.g., layer reordering) can be reconfigured at runtime to maintain QoS under shifting conditions [17]. This approach, combined with optimal orchestration policies, has the potential to cater to the increasingly demanding AI inference workloads in future AIaaS 6G-enabled networks and edge environments.

C. Key Design Goals of Adaptive LFM Split Inference

Practical deployments remain constrained by *static* or *coarse-grained* orchestration mechanisms [4], [18]. Thus, current solutions cannot adapt to shifting network or compute conditions, leading to latency spikes, resource imbalances, and potential SLA and QoS violations [19]. Meanwhile, next-generation 6G network architectures will further exacerbate the complexity of distributing large-scale inference workloads across heterogeneous edge topologies to support various commercial and operational AIaaS applications [1]. Hence, an *adaptive split inference* framework will be required that:

1) dynamically reconfigures the partition of LFM layers among edge and cloud compute nodes,
2) exploits real-time profiling of resource availability,
3) preserves data privacy by keeping computation local, and
4) ensures consistent, QoS/SLA-compliant performance.

In the next section, we propose a novel orchestration method that closes this gap by intelligently managing LFM inference across edge compute infrastructures.

III. Proposed Adaptive Orchestration Framework

We propose an *adaptive split inference orchestration* framework that dynamically manages LFM partitions between nodes. Fig. 2 depicts a possible realization of this framework in a 5G/6G-MEC deployment, including components for monitoring, decision-making, model partitioning, and reconfiguration. We outline a detailed reference architecture as follows.

A. Reference Architecture

Our framework orchestrates on-demand allocation and reallocation of LFM partitions under evolving operational conditions via the following core modules:

1) **Monitoring & Capacity Profiling (CP):** Collects real-time metrics from edge nodes and the network such as CPU/GPU utilization, memory usage, bandwidth, and latency. These metrics guide the orchestrator in partition placement and corresponding re-splitting decisions.
2) **Adaptive Orchestrator (AO):** Acts as the decision-making engine by evaluating whether to:
   - *Keep* the current split (no changes).
   - *Redistribute* sub-splits across underutilized nodes.
   - *Fully re-split* the model.

   These decisions are informed by constraints like node capacity, privacy requirements, and expected QoS.
3) **Split Revision (SR):** Implements the logic to re-partition the LFM and may use heuristic, rule-based, or learning-based strategies to identify improved splits.

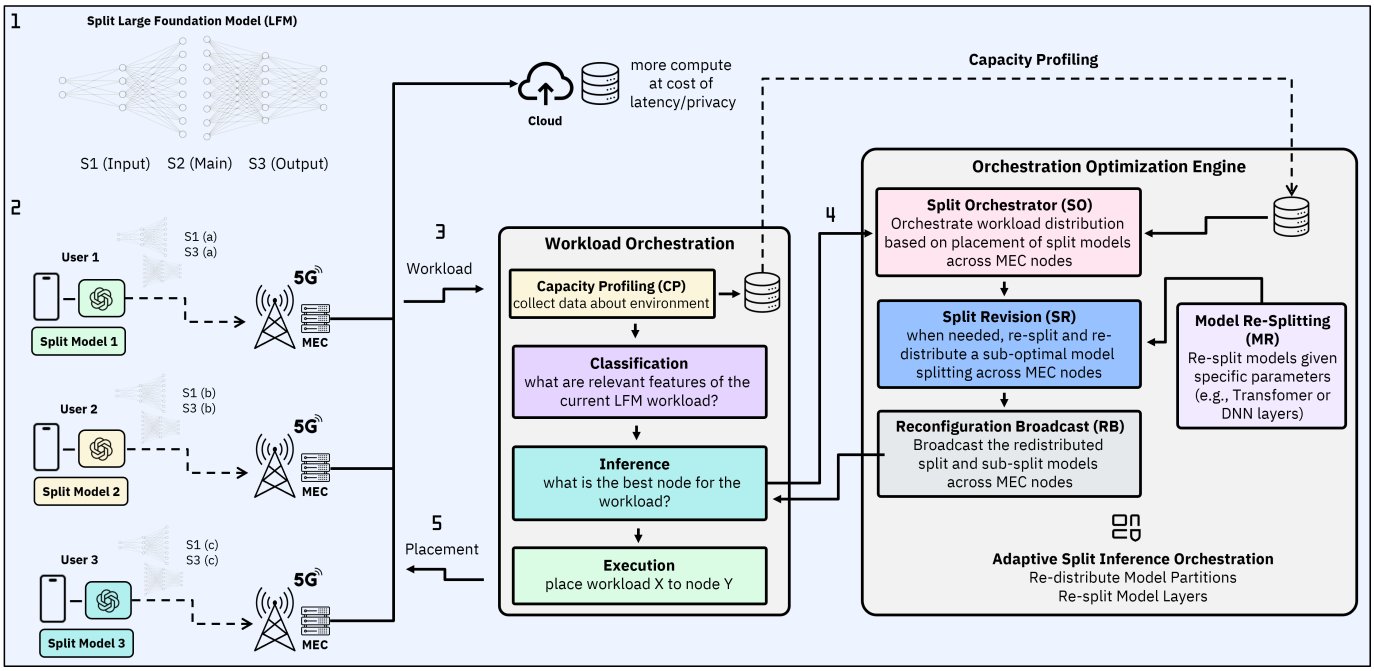

Fig. 2. Reference architecture of the proposed adaptive split inference orchestration. Sub-split models (S1, S2, S3) are deployed across edge/cloud nodes, while a central orchestrator, guided by real-time capacity profiling, re-splits and reconfigures workloads on demand to meet QoS and privacy constraints.

4) **Reconfiguration Broadcast (RB):** Broadcasts new model partitions or sub-partitions (S_{1a}, S_{1b}, etc.) to the selected nodes and updates local or remote orchestrators.

Our approach dynamically adapts split inference to fluctuating conditions while maintaining strict QoS and privacy requirements by combining these modules. The next subsections formalize the system model, define constraints, and describe the orchestration workflow for LFMs in detail.

B. Notation and System Model

We define key terminologies and orchestration concepts that underlie our adaptive split inference framework as follows.

- **Compute Nodes.** Let $\mathcal{N} = \{1, 2, \ldots, n\}$ denote the set of n edge nodes, and let c refer to a cloud node. Each node $j \in \mathcal{N} \cup \{c\}$ has capacities for inference at time t:

$$\text{CP}(n_j, t) = \{\text{CPU}_j(t), \text{GPU}_j(t), \text{Mem}_j(t), \text{NetCap}_j(t)\}. \quad (1)$$

- **Partitioning.** Consider an LFM $\mathcal{M}$ segmented into k consecutive model layers:

$$S = \{S_1, S_2, \ldots, S_k\}. \quad (2)$$

Typically, S_1 handles raw (potentially private) data and S_k generates the final outputs. Intermediate segments $S_2, \ldots, S_{k-1}$ often encompass the bulk of computation. A three-split example $\{S_1, S_2, S_3\}$ might place S_1, S_3 on a local edge node (for privacy where user data is translated into/from features) and offload the compute-intensive S_2 to a more capable node. Depending on the specific LFM architecture, splits may either be configured as self-contained building blocks or individual layers (e.g., from deep or convolutional neural networks) [4].

- **Inference Requests.** Inference tasks arrive as requests $\{r_1, r_2, \ldots\}$, each with an associated workload $\mathcal{W}_r$. At a high level, each request utilizes the same partitions $\{S_1, \ldots, S_k\}$, but may require separate scheduling decisions depending on QoS constraints or capacity.
- **Decision Variables.** For convenience, we define a binary placement matrix $\mathbf{x} = [x_{i,j}]$, where $x_{i,j} = 1$ indicates partition S_j is assigned to node n_i and $x_{c,j} = 1$ indicates assignment to the cloud node c. Each column corresponds to a partition and each row to a node in $\mathcal{N} \cup \{c\}$. When multiple requests are considered, either the same $\mathbf{x}$ can be reused if the system enforces a single partition, or a time/index extension can be used (e.g., $\mathbf{x}_r$ for each r).

With that, we define the optimization objective as follows.

Objective Function. We aim to minimize the high-level cost:

$$\Phi(\mathbf{x}, \mathcal{C}(t)) = \alpha\,\mathcal{L}\big(\mathbf{x}, \mathcal{C}(t)\big) + \beta\,\mathcal{U}\big(\mathbf{x}, \mathcal{C}(t)\big) + \gamma\,\mathcal{P}\big(\mathbf{x}, \mathcal{C}(t)\big),$$

where:

- $\mathcal{L}$ measures inference latency, including data transfer,
- $\mathcal{U}$ captures resource usage imbalance or node overload,
- $\mathcal{P}$ penalizes privacy violations (e.g., placing sensitive partitions on untrusted nodes), and
- $\alpha, \beta, \gamma \geq 0$ weigh the relative importance of latency, resource usage, and privacy, respectively.

Here, $\mathcal{C}(t)$ encapsulates the system state at time t, including node capacities, network bandwidths, and any QoS/SLA requirements. In scenarios with concurrent requests, Φ can be

extended to the sum or average cost across all requests. In addition, to ensure valid assignments, we impose constraints:

1) **Unique Assignment.** Each partition S_j must be placed on exactly one node:

$$\sum_{i \in \mathcal{N}} x_{i,j} \; + \; x_{c,j} = 1, \quad \forall j \in \{1, \ldots, k\}. \tag{3}$$

2) **Capacity Limits.** For each node $n_i \in \mathcal{N}$, the sum of resource loads from its assigned partitions cannot exceed that node's capacity:

$$\sum_{j=1}^{k} \text{load}\big(S_j\big)\, x_{i,j} \;\; \leq \;\; \text{capacity}(n_i, t). \tag{4}$$

3) **Privacy Constraints.** Partitions handling sensitive data (e.g., S_1) must remain on trusted nodes:

$$x_{i,j} = 0, \quad \text{if } n_i \notin \text{trustedSet} \wedge \big(S_j \text{ is privacy-critical}\big). \tag{5}$$

Further, if LFM layer boundaries can be modified (e.g., subdividing S_2 into $\{S_{2a}, S_{2b}\}$, as for example in neural network layers, transformer embeddings, attention layers, etc.), we may treat the set of partitions S itself as part of the optimization. Herewith, we define the split revision as follows.

Split Revision (SR). Let Ω denote the set of all valid splitting schemes. The orchestrator aims to solve

$$\min_{S \in \Omega,\, \mathbf{x}} \quad \Phi\big(\mathbf{x}, S, \mathcal{C}(t)\big) \tag{6}$$

to find an optimal split S^* and assignment $\mathbf{x}^*$ that minimizes the overall cost subject to the constraints above. This allows partitions and assignments to adapt dynamically to shifts in resource availability and workload demands.

C. Orchestration Workflow

Alg. 1 outlines the main orchestration steps. The workflow begins by deploying a *baseline* partition (e.g., $\{S_1, S_2, S_3\}$) among a set of nodes. The system then continuously monitors resource usage and performance metrics to trigger adjustments.

1) **Initial Deployment.** Perform a static partitioning of the model based on coarse performance estimates (e.g., place S_1, S_3 locally for privacy and put S_2 on a cloud node c).
2) **Continuous Monitoring.** The CP module collects real-time metrics $\text{CP}(n_j, t)$ and calculates an *environment state* $\mathbf{E}(t)$ that captures fluctuations in node utilization.
3) **Adaptive Decisions.** Based on the updated system states $\mathcal{C}(t), \mathbf{E}(t)$, the adaptive orchestrator continuously evaluates whether to *keep the current split*, *redistribute sub-splits* (reassigning some partitions S_j from node n_i to $n_{i'}$ by adjusting $\mathbf{x}$ without altering the partition boundaries), or *perform full re-splitting* (to obtain a better partition set S^* via the SR module if incremental changes are insufficient or new privacy constraints arise). More formally:

Algorithm 1: Adaptive Split Orchestration Workflow

Input: (i) Initial partitioning $\{S_1, \ldots, S_P\}$, (ii) baseline mapping d_0, (iii) monitoring intervals Δt, (iv) trigger-threshold vector $\Theta = \{L_{\max}, U_{\max}, B_{\min}, T_{\text{cool}}\}$
Initialize: Deploy baseline split $(S_1, \ldots, S_P)$ across nodes as per d_0.
Set $t_{\text{last}} \leftarrow -\infty$.
for each monitoring cycle $t \leftarrow 0, \Delta t, 2\Delta t, \ldots$ **do**
  Collect environment metrics $\mathbf{E}(t)$ via Monitoring & CP.
  reconf $\leftarrow$ `ShouldReconfigure`$(\mathbf{E}(t), \Theta)$.
  if *(*trigger condition is met, e.g., high latency, node overload, etc.*)* ***and*** *reconf* **then**
    Evaluate feasible mappings $\{d'\}$ given current partitions.
    Optionally call Model Re-Splitting to produce new partitions $\{S_i^*\}$.
    Determine best mapping $\hat{d} = \arg\min_{d'} \mathcal{C}(d')$.
    if $\hat{d} \neq d_t$ ***and*** $t - t_{\text{last}} \geq T_{cool}$ **then**
      Broadcast reconfiguration to all affected nodes via RB.
      $t_{\text{last}} \leftarrow t; \quad d_{t+\Delta t} \leftarrow \hat{d}$.
    end
  Resume inference under current assignment $d_{t+\Delta t}$.
end

- The adaptive orchestrator evaluates whether the current partition mapping d_t remains optimal under $\mathbf{E}(t)$. For each request r, the orchestrator checks:

$$\mathcal{C}(d_t) \;\overset{?}{\leq}\; \mathcal{C}(d') \quad \forall \text{ feasible } d'. \tag{7}$$

- If needed, the SR module modifies the set of partitions $\{S_1, \ldots, S_P\}$ (e.g., subdividing a large block S_2 into new split configurations $\{S_{2a}, S_{2b}\}$), i.e.

$$\hat{d} \;=\; \underset{d \,\in\, \mathcal{D}(\text{new splits})}{\text{argmin}} \;\; \mathcal{C}(d), \tag{8}$$

  subject to constraints (e.g., compute, network, privacy).

4) **Reconfiguration Broadcast (RB).** Once a decision is made, the *RB* module disseminates the updated assignment $\mathbf{x}^*$ or partition set S^* to relevant nodes, ensuring the new configuration is deployed consistently.
5) **Execution.** Inference resumes with the updated partition assignment $\hat{d}$. The orchestrator continues to monitor performance, forming a feedback loop and allowing the system to adapt further as conditions evolve.

Trigger Conditions and Decision Logic. Table I summarizes the runtime metrics that feed the function *ShouldReconfigure*(E(t),Θ) in Alg. 1. A reconfiguration is invoked if *any* of the following holds for a monitoring window of length Δt:

1) **Latency threshold.** The exponentially weighted moving average (EWMA) of end-to-end inference latency $\bar{L}(t, \Delta t)$ exceeds $L_{\max}$, i.e., $\bar{L}(t, \Delta t) > L_{\max}$.
2) **Utilisation threshold.** The maximum node utilization exceeds $U_{\max}$, i.e., $\max_{n \in \mathcal{N}} U_n(t) > U_{\max}$.
3) **Bandwidth drop.** The minimum available bandwidth drops below $B_{\min}$, i.e., $\min_{(i,j) \in \mathcal{L}} B_{ij}(t) < B_{\min}$.

TABLE I
MONITORED METRICS AND DEFAULT TRIGGER THRESHOLDS.

Metric	Symbol	Empirical Value
EWMA latency	$L_{\max}$	150 ms
GPU/CPU utilization	$U_{\max}$	0.85
Available link bandwidth (edge-to-edge)	$B_{\min}$	50 Mbps
Time-to-reconfigure cool-down	T_{cool}	30 s

If multiple triggers fire simultaneously, the system first attempts *placement migration*. If that cannot meet all constraints, the *split-revision* module is invoked. Reconfigurations are further rate-limited by T_{cool} to prevent thrashing.

In practice, such an orchestration loop can be integrated into existing platforms (e.g., extending Kubernetes with a custom controller for re-splitting). Partitioning decisions may then rely on traditional heuristics (e.g., rule-based or greedy approaches) or adopt learning-based schemes (e.g., reinforcement learning) to continuously refine splitting strategies [20], [21]. We leave the investigation of optimal strategies to future work.

D. Privacy and Security Considerations

A core feature in split inference is the preservation of data privacy by ensuring critical or sensitive operations remain on a trusted device or node. Thus, our framework permits a *selective local execution*. For example, some LFM blocks, especially those close to the input layer, may handle raw personal or private data. By design, these partitions can be configured to remain on the client device or a trusted edge node. Formally, if S_i handles private data, we require that

$$d_t(i) \in \mathcal{N}_{\text{trusted}} \quad \forall t, \tag{9}$$

where $\mathcal{N}_{\text{trusted}} \subseteq \mathcal{N} \cup \{c\}$ is the set of *trusted nodes*. Corresponding LFM splits can be obtained according to model architecture, compute resources, and privacy requirements (e.g., measured as layer depth) [20]. By leveraging partial LFM layer splits, our orchestration framework thus inherently supports privacy-preserving inference at the edge, ensuring that sensitive data never leaves a trusted domain. Furthermore, our framework can be extended with secure encryption and transmission protocols to safeguard intermediate activations (e.g., outputs from S_1 that serve as inputs to S_2) against malicious entities such as jammers and eavesdroppers [22]. Beyond inference, our framework readily supports post-hoc safety alignment methods such as SafeMERGE [23], which exchange or merge individual model layers with safe counterparts to make them resilient against fine-tuning attacks. By facilitating these layer adaptations, our architecture becomes highly adaptable to a broad spectrum of defense mechanisms.

IV. EXPECTED PERFORMANCE RESULTS

We evaluate our adaptive orchestration framework for a simulated 5G-MEC urban scenario based on figures reported across four public studies [15], [24]–[26], where we derive *expected KPIs* by combining (i) the ETSI MEC latency model for processing, queuing, and fronthaul/backhaul, as well as (ii) empirical improvements reported by recent split inference works [15], [17]. We choose trigger thresholds as in Table I. While we acknowledge that a comparison with baselines like [15] and [24] is valuable, it remains challenging as they lack runtime graph re-splitting capabilities. Thus, we leave comprehensive cross-framework benchmarking to future work.

TABLE II
EXPECTED STEADY-STATE KPIS FOR A 10S MONITORING WINDOW.

Backhaul (Mb/s)	Static-Split Latency (ms)	Adaptive Latency (ms)	Δ Latency	Throughput (× Baseline)	GPU Util.
20	500	200	–60%	2.1×	92%
50	320	150	–53%	2.0×	90%
100	230	120	–48%	1.9×	88%
200	180	110	–39%	1.8×	86%

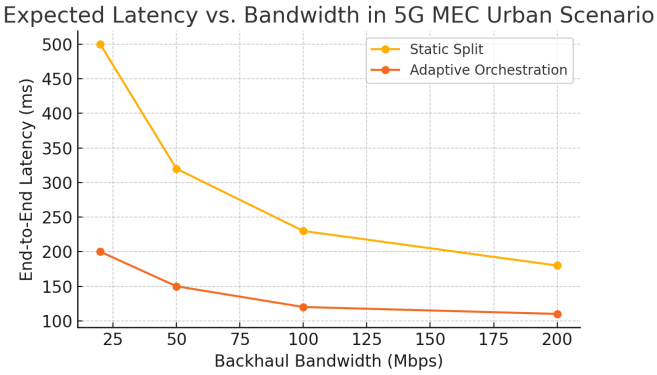

Fig. 3. Expected latency vs. bandwidth availability in our 5G MEC urban scenario. Adaptive split inference orchestration results in lower total latency.

a) Scenario & Methodology: Three MEC nodes, each with an NVIDIA A100 40GB GPU and cloud compute capabilities, serve intermittent LFM inference requests for Llama3 models of 8B parameter size [27]. The baseline uses a static $\{S_1, S_2, S_3\}$ split, whereas our adaptive orchestrator may migrate segments or re-split S_2 on QoS triggers. We sweep backhaul bandwidths $\{20, 50, 100, 200\}$ Mb/s and compute:

$$\text{latency}_{\text{static/adaptive}} = \underbrace{T_{\text{proc}} + T_{\text{queue}}}_{\text{ETSI MEC model}} + T_{\text{tx}}(\text{bandwidth}), \tag{10}$$

b) Key Metrics: In our analysis, we focus on latency as the key performance indicator (KPI). Table II summarizes expected steady-state KPIs averaged over a 10s monitoring window, where deltas inherit margins from related works [24]–[26] and scale with the analytical latency model. Fig. 3 plots the corresponding end-to-end latency versus backhaul bandwidth. In general, we observe the following improvements:

- **Bandwidth.** Static-split latency falls sharply with bandwidth, whereas adaptive orchestration flattens the curve by migrating or re-splitting to avoid link chokepoints.
- **Diminishing returns.** Above ∼100 Mb/s, transport ceases to dominate and gains come from GPU load balancing, narrowing the static-adaptive gap.

- **QoS guarantees.** The 150 ms URLLC bound is met across all loads *only under the adaptive scheme*, highlighting its robustness for V2X and XR services.
- **Resource utilization.** Adaptive re-slicing keeps GPU usage near 90%, avoiding stragglers and over-provisioning.

Adaptive split inference thus consistently outperforms any static configuration. The small overhead of monitoring ($\leq$10 ms per cycle) and graph rewiring are amortized by hundreds of ms saved per request, yielding a net performance gain an order of magnitude larger than the orchestration cost. Consequently, adaptive split inference emerges as the decisive design choice in heterogeneous, bandwidth-variable edge networks.

V. Conclusions

This study has introduced an adaptive split inference orchestration framework designed to dynamically manage LFM partitions across heterogeneous edge nodes, establishing a foundation for real-time, QoS-aware, and privacy-preserving AI inference in edge computing environments. A key innovation of our approach is treating both model partitioning and node placement as dynamic, runtime-resolved variables, rather than fixed pre-conditions. This design directly addresses the variability in layer-splitting complexity across different LFMs. By operating on the underlying computational graph, our framework remains generalizable across diverse architectures without requiring model-specific redesigns. This unified optimization ensures the system can support latency-sensitive inference and data locality under the volatility of 6G edge environments, offering a scalable substrate that remains extensible to future inference graphs and scheduling objectives. Further, our framework can be seamlessly integrated with existing orchestration platforms at low cost due to its modular architecture, while maintaining extensibility for future AI-driven optimizations. Our proposed orchestration model thus optimizes performance, enhances resource efficiency, and fortifies privacy preservation, aligning with emerging objectives in the development of future AI-native 6G and beyond networks.

References

[1] H. Zhou, C. Hu, Y. Yuan, Y. Cui, Y. Jin, C. Chen, H. Wu, D. Yuan, L. Jiang, D. Wu, X. Liu, C. Zhang, X. Wang, and J. Liu, "Large Language Model (LLM) for Telecommunications: A Comprehensive Survey on Principles, Key Techniques, and Opportunities," *IEEE Communications Surveys & Tutorials*, pp. 1–1, 2024.

[2] B. Li, Y. Jiang, V. Gadepally, and D. Tiwari, "LLM Inference Serving: Survey of Recent Advances and Opportunities," *arXiv preprint arXiv:2407.12391*, 2024.

[3] J. Karjee, P. Naik S, K. Anand, and V. N. Bhargav, "Split computing: DNN Inference Partition with Load Balancing in IoT-Edge Platform for Beyond 5G," *Measurement: Sensors*, vol. 23, p. 100409, 2022.

[4] J. Karjee, K. Anand, V. N. Bhargav, P. S. Naik, R. B. V. Dabbiru, and N. Srinidhi, "Split Computing: Dynamic Partitioning and Reliable Communications in IoT-Edge for 6G Vision," in *2021 8th International Conference on Future Internet of Things and Cloud (FiCloud)*, 2021.

[5] I. Vasireddy, G. Ramya, and P. Kandi, "Kubernetes and Docker Load Balancing: State-of-the-Art Techniques and Challenges," *International Journal of Innovative Research in Engineering and Management*, vol. 10, no. 6, pp. 49–54, 2023.

[6] R. Xu, N. Baracaldo, and J. Joshi, "Privacy-Preserving Machine Learning: Methods, Challenges and Directions," *arXiv preprint arXiv:2108.04417*, 2021.

[7] K. B. Letaief, Y. Shi, J. Lu, and J. Lu, "Edge Artificial Intelligence for 6G: Vision, Enabling Technologies, and Applications," *IEEE journal on selected areas in communications*, vol. 40, no. 1, pp. 5–36, 2021.

[8] C. Carrión, "Kubernetes Scheduling: Taxonomy, Ongoing Issues and Challenges," *ACM Computing Surveys*, vol. 55, no. 7, pp. 1–37, 2022.

[9] J. Duan, S. Zhang, Z. Wang *et al.*, "Efficient Training of Large Language Models on Distributed Infrastructures: A Survey," 2024. [Online]. Available: https://arxiv.org/abs/2407.20018

[10] J. Lang, Z. Guo, and S. Huang, "A Comprehensive Study on Quantization Techniques for Large Language Models," in *IEEE ICAIRC*, 2024.

[11] Z. Zhou, X. Ning, K. Hong, T. Fu, J. Xu, S. Li, Y. Lou, L. Wang, Z. Yuan, X. Li *et al.*, "A Survey on Efficient Inference for Large Language Models," *arXiv preprint arXiv:2404.14294*, 2024.

[12] W. Saad, O. Hashash, C. K. Thomas, C. Chaccour, M. Debbah, N. Mandayam, and Z. Han, "Artificial General Intelligence (AGI)-Native Wireless Systems: A Journey Beyond 6G," 2024. [Online]. Available: https://arxiv.org/abs/2405.02336

[13] T. Mohammed, C. Joe-Wong, R. Babbar, and M. D. Francesco, "Distributed Inference Acceleration with Adaptive DNN Partitioning and Offloading," in *IEEE INFOCOM 2020 - IEEE Conference on Computer Communications*, 2020, pp. 854–863.

[14] Z. Lin, G. Qu, Q. Chen, X. Chen, Z. Chen, and K. Huang, "Pushing Large Language Models to the 6G Edge: Vision, Challenges, and Opportunities," *arXiv preprint arXiv:2309.16739*, 2023.

[15] M. Zhang, X. Shen, J. Cao, Z. Cui, and S. Jiang, "EdgeShard: Efficient LLM Inference via Collaborative Edge Computing," *IEEE Internet of Things Journal*, pp. 1–1, 2024.

[16] N. Hudson, H. Khamfroush, M. Baughman, D. E. Lucani, K. Chard, and I. Foster, "QoS-Aware Edge AI Placement and Scheduling with Multiple Implementations in FaaS-Based Edge Computing," *Future Generation Computer Systems*, vol. 157, pp. 250–263, 2024.

[17] Y. Chen, R. Li, X. Yu, Z. Zhao, and H. Zhang, "Adaptive Layer Splitting for Wireless LLM Inference in Edge Computing: A Model-Based Reinforcement Learning Approach," 2024. [Online]. Available: https://arxiv.org/abs/2406.02616

[18] L. Zhou, H. Wen, R. Teodorescu, and D. H. Du, "Distributing Deep Neural Networks with Containerized Partitions at the Edge," in *2nd USENIX Workshop on Hot Topics in Edge Computing*, 2019.

[19] Y. Li, X. Wang, X. Gan, H. Jin, L. Fu, and X. Wang, "Learning-Aided Computation Offloading for Trusted Collaborative Mobile Edge Computing," *IEEE Transactions on Mobile Computing*, vol. 19, no. 12, pp. 2833–2849, 2019.

[20] X. Li and S. Bi, "Optimal AI Model Splitting and Resource Allocation for Device-Edge Co-Inference in Multi-User Wireless Sensing Systems," *IEEE Transactions on Wireless Communications*, vol. 23, no. 9, pp. 11 094–11 108, 2024.

[21] S.-Y. Lien, C.-H. Yeh, and D.-J. Deng, "Optimum Splitting Computing for DNN Training Through Next Generation Smart Networks: A Multi-Tier Deep Reinforcement Learning Approach," *Wireless Networks*, vol. 30, no. 3, pp. 1737–1751, 2024.

[22] A. Djuhera, V. C. Andrei, X. Li, U. J. Mönich, H. Boche, and W. Saad, "R-SFLLM: Jamming Resilient Framework for Split Federated Learning with Large Language Models," 2024. [Online]. Available: https://arxiv.org/abs/2407.11654

[23] A. Djuhera, S. R. Kadhe, F. Ahmed, S. Zawad, and H. Boche, "SafeMERGE: Preserving Safety Alignment in Fine-Tuned Large Language Models via Selective Layer-Wise Model Merging," 2025. [Online]. Available: https://arxiv.org/abs/2503.17239

[24] G. Zhang, W. Guo, Z. Tan, and H. Jiang, "AMP4EC: Adaptive Model Partitioning Framework for Efficient Deep Learning Inference in Edge Computing Environments," 2025. [Online]. Available: https://arxiv.org/abs/2504.00407

[25] S. Tuli, G. Casale, and N. R. Jennings, "SplitPlace: AI Augmented Splitting and Placement of Large-Scale Neural Networks in Mobile Edge Environments," 2022. [Online]. Available: https://arxiv.org/abs/2205.10635

[26] A. Mudvari, A. Vainio, I. Ofeidis, S. Tarkoma, and L. Tassiulas, "Adaptive Compression-Aware Split Learning and Inference for Enhanced Network Efficiency," 2024. [Online]. Available: https://arxiv.org/abs/2311.05739

[27] A. Grattafiori, A. Dubey, A. Jauhri, A. Pandey, A. Kadian, A. Al-Dahle, A. Letman, A. Mathur, A. Schelten, A. Vaughan *et al.*, "The Llama 3 Herd of Models," *arXiv preprint arXiv:2407.21783*, 2024.