applied sciences

Article

CoMEx: Continual Mixture of Experts for Fast Policy Adaptation in RAN Slicing

Xian Mu [1,2], Mingzhu Liu [2], Yao Xu [1,3] and Dagang Li [1,4,*]

[1] School of Computer Science and Engineering, Macau University of Science and Technology, Macau 999078, China; xianmu@nut.edu.cn (X.M.); xuyao@fynu.edu.cn (Y.X.)
[2] College of Computer Information and Engineering, Nanchang Institute of Technology, Nanchang 330044, China; 2012043@nut.edu.cn
[3] School of Economics, Fuyang Normal University, Fuyang 236037, China
[4] Zhuhai M.U.S.T. Science and Technology Research Institute, Zhuhai 519000, China
[*] Correspondence: dgli@must.edu.mo

Abstract

Network slicing is a cornerstone of 5G/6G vertical services, yet practical deployments require mobile network operators (MNOs) to adjust slice service level agreement (SLA) weights based on quality of experience (QoE), causing rapid non-stationary objective changes that can destabilize deep reinforcement learning (DRL) slicing policies and necessitate retraining. This paper proposes Continual Mixture of Experts (CoMEx) for fast policy adaptation. CoMEx pre-trains and freezes multiple expert policies under diverse SLA preferences, explicitly appends the SLA weight vector to observations, and trains a DRL-based gating network to fuse expert actions at the step level for fast adaptation to unseen SLA configurations. To broaden coverage without degrading existing experts, CoMEx further incorporates a masked expert expansion mechanism that incrementally adds new experts and fine-tunes the gate. Step-level DRL gating demonstrates superior generalization in RAN slicing, attaining a mean score of 78.95 under unseen SLA weights—surpassing episode-level and supervised gating by 2.40% and 27.67%, respectively. Moreover, CoMEx's extensibility is highlighted by a 7.08% performance boost (reaching 84.54) upon the addition of a fourth expert. Such results confirm the framework's capacity for timely and robust policy adaptation in non-stationary SLA environments.

Keywords: radio access network (RAN) slicing; Mixture of Experts (MoE); deep reinforcement learning; SLA satisfaction; incremental learning

Academic Editor: Christos Bouras

Received: 15 January 2026
Revised: 4 February 2026
Accepted: 10 February 2026
Published: 12 February 2026

1. Introduction

Radio access network (RAN) slicing enables mobile network operators (MNOs) to virtualize shared physical infrastructure into multiple logical slices [1–3], each tailored to distinct service requirements [4], as illustrated in Figure 1. In practical deployments, slice level service-level agreement (SLA) priorities are not static: MNOs frequently adjust slice weightings to maintain acceptable overall quality of experience (QoE) under fluctuating traffic and network conditions [5]. Such changes shift the optimization objective and can make previously trained resource-allocation agents suboptimal, often triggering retraining.

However, retraining an DRL agent for RAN slicing is costly and operationally undesirable [6]. First, training typically requires substantial interaction data and computational resources, which is incompatible with real-time SLA adjustments. Second, online exploration during retraining may degrade service quality [7]. Third, SLA-induced non-stationarity further destabilizes convergence. Therefore, a key challenge is to design a slicing policy that can rapidly adapt to new SLA weightings while maintaining stable performance, without full retraining.

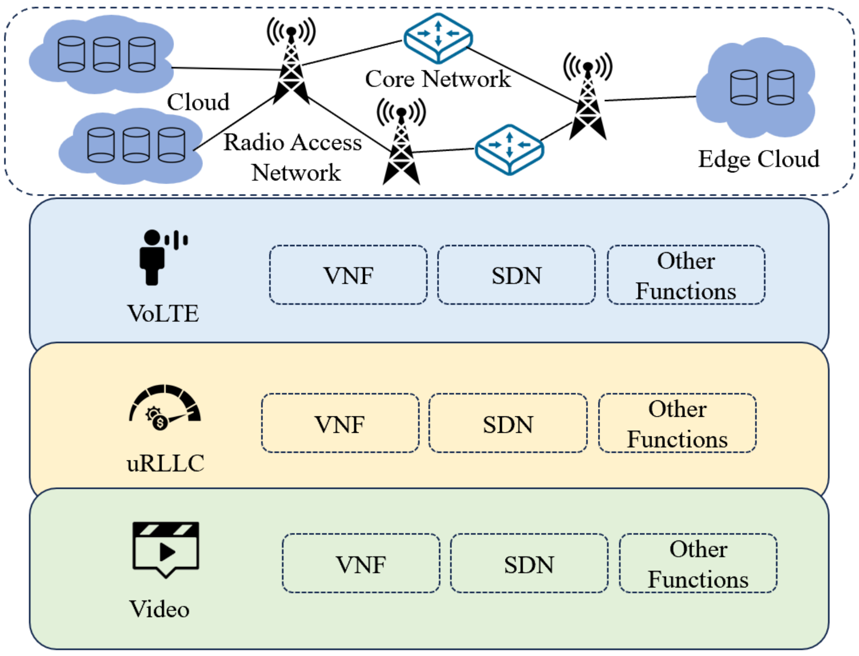

Figure 1. RAN slicing architecture with Virtualized Network Functions for Heterogeneous Services.

To address this challenge, we propose Continual Mixture of Experts (CoMEx) for fast policy adaptation in RAN slicing. CoMEx maintains a frozen expert pool and trains an SLA-aware gating network that fuses expert actions conditioned on the current network state and the target SLA weight vector. It further supports continual expert expansion by adding new experts and incrementally updating the gate, enabling long-term evolution without retraining existing experts, as illustrated in Figure 2. The main contributions of this paper are summarized as follows:

- An SLA-aware policy adaptation framework. We embed SLA weightings into the observation space and employ a step-level gating network to dynamically fuse frozen expert policies, enabling zero-shot and rapid adaptation to previously unseen SLA configurations.
- Adaptive DRL-based gating optimization. We train the gating network via reinforcement learning to make end-to-end expert-fusion decisions conditioned on real-time network states and SLA objectives, improving robustness under time-varying traffic and out-of-distribution SLA weights.
- Masked continual expert expansion. We introduce a masked expert-pool expansion mechanism that supports incremental addition of new experts, enabling continual improvement without retraining existing experts and mitigates the risk of catastrophic forgetting at the mechanism level.

The rest of the paper is organized as follows. Section 2 reviews the related work. Section 3 formulates the system model and problem formulation. Section 4 presents the proposed CoMEx framework. Section 5 reports experimental and ablation results. Section 6 discusses limitations. Finally, Section 7 concludes the paper and discusses future work.

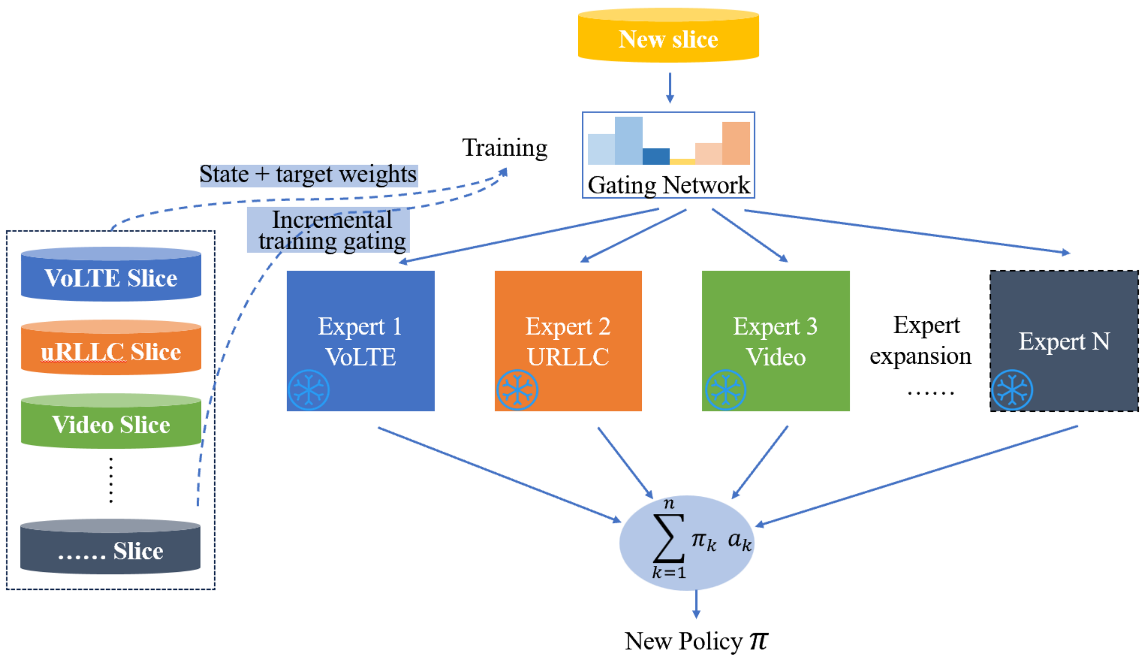

Figure 2. Illustration of CoMEx framework for Dynamic RAN Slicing.

2. Related Work

In network slicing scenarios, conventional resource allocation strategies typically rely on complex mathematical models or heuristic algorithms [8,9], which often struggle to cope with the highly dynamic and strongly non-stationary characteristics of 5G/6G networks [10,11]. Owing to its ability to learn near-optimal policies through interaction with the environment, deep reinforcement learning (DRL) has been widely adopted to address the NP-hard resource scheduling problem in RAN slicing [11,12]. For instance, an intelligent slicing scheduling framework based on the asynchronous advantage actor–critic (A3C) algorithm was proposed to improve resource multiplexing gains while maintaining slice isolation [11]. To further enforce SLAs, several studies introduced constrained reinforcement learning (RL), which explicitly incorporates SLA constraints to reduce violation rates [13]. In addition, model-driven RL approaches leverage kernel-based classifiers and self-evaluation mechanisms to achieve higher sample efficiency and improved safety during online learning [14].

To improve scalability while maintaining constant computational overhead, sparsely activated MoE models have become increasingly prevalent [15]. In natural language processing, adaptive gating mechanisms have been used to dynamically adjust the number of activated experts according to token complexity [15]. In the RL context, MoE has been leveraged to mitigate parameter conflicts in multi-task learning. For example, the M3DT framework combines Decision Transformer with MoE and achieves strong performance at the scale of 160 tasks [16]. In continual learning (CL), MoE has been shown to effectively alleviate catastrophic forgetting, where the gating network preserves prior knowledge by routing different tasks to specialized experts [17]. However, most existing MoE studies in wireless networks adopt fixed gating strategies, and the capability of gating mechanisms to rapidly adapt to time-varying SLA weightings remains under-explored [18].

In RANs, the slow convergence of RL agents remains a major obstacle to practical deployment [12]. Transfer learning accelerates training in the target domain by reusing expert policies learned in a source domain [19]. Teacher–Assistant distillation has also been explored to improve policy transfer efficiency under changing slicing objectives [20]. To address the non-stationarity induced by MNOs adjusting SLA weightings, prior work has proposed predictive policy transfer, which selects and reuses the most suitable expert policy based on the Euclidean distance between SLA weight vectors [5]. In addition, prompt-based architectures enable few-shot task adaptation without parameter fine-tuning by feeding a small set of trajectories as prompts into a Transformer [21]. Hybrid transfer learning strategies that combine policy reuse and distillation further improve the stability and efficiency of online exploration [6].

Classical CL approaches include regularization-based methods that protect critical parameters and replay-based methods that store historical samples [17]. Recent studies show that MoE models can naturally select experts via adaptive routing, and that experts can learn more accurate feature representations through mutual distillation [22]. In multi-agent systems, combining a global critic with locally observed information helps maintain cooperative performance under dynamic environments [23].

While RL-driven slicing and MoE architectures have advanced, existing methods still face limitations under frequent SLA re-weighting by MNOs. Policy transfer methods reuse pre-trained policies but lack a w-conditioned fusion rule and rely on retraining for large shifts in preferences. Moreover, while recent 2023–2025 advances in transfer learning and meta-RL [12,19–21] significantly accelerate adaptation, they typically necessitate online parameter fine-tuning or assume domain-specific environment migration. These paradigms differ from our focus on zero-shot adaptation to rapid SLA weight drift under strict "no-retraining" operational constraints. Gating-based fusion methods typically use offline or heuristic routing or episode-level dispatching, which are less responsive to real-time traffic dynamics and limit continual expansion. There is a need for a mechanism that addresses these gaps by learning step-level, SLA-conditioned expert fusion in closed-loop RL for fast adaptation to unseen $\mathbf{w}$, and enabling incremental expert expansion through gated updates.

3. System Model and Problem Formulation

This section presents the system model for dynamic RAN slicing and formulates the SLA-aware resource allocation task as a partially observable Markov decision process (POMDP).

3.1. Slicing Resources

RAN slicing enables the shared RAN infrastructure to be logically segmented into multiple service-specific slices, each supporting heterogeneous traffic profiles under different QoS constraints [4]. We consider a RAN slicing system with N slices and adopt a time-window-based resource allocation mechanism. Time is partitioned into discrete scheduling steps, each corresponding to a fixed-length scheduling window (comprising multiple transmission time intervals (TTIs)). At the beginning of each scheduling step, the RL agent makes a resource allocation decision, which remains unchanged for all TTIs within the corresponding scheduling window.

Let R_{total} denote the total amount of available radio resources per TTI, measured in physical resource blocks (PRBs). The resource allocation decision at each step is represented by

$$\mathbf{r} = [r_1, r_2, \ldots, r_N], \tag{1}$$

where r_i denotes the number of PRBs allocated to slice i per TTI in the current scheduling window.

The system is subject to the following resource constraint:

$$\sum_{i=1}^{N} r_i = R_{\text{total}}. \tag{2}$$

In each scheduling window, the agent outputs a continuous action vector $\mathbf{a} = [a_1, a_2, \ldots, a_N]$. This vector is normalized to obtain resource allocation weights and is then mapped to integer-valued PRBs allocations according to the total resource constraint. The allocation action remains fixed throughout the scheduling window and governs packet transmissions over all TTIs within that window, as illustrated in Figure 3. This step-based resource allocation mechanism enables the system to capture scheduling effects across TTIs while maintaining tractable decision-making at the agent level. To avoid introducing unrealistic prior information about future arrivals, we model RAN slicing resource allocation as a POMDP. At each scheduling window, the agent only observes statistics of the current queue backlog, while the newly arriving traffic within the window acts as an unobserved disturbance that affects system evolution and the resulting rewards. This formulation more closely matches practical online scheduling and provides a more stringent test of policy robustness.

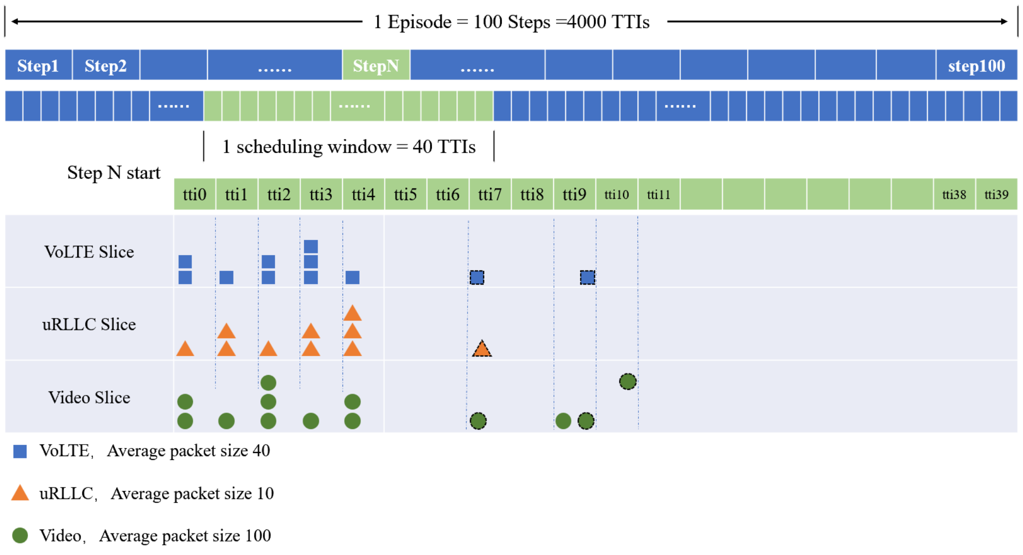

Figure 3. Illustration of Window-Based Resource Allocation and Cross-TTI Packet Scheduling Mechanism.

3.2. Service Level Agreement

In practical RAN slicing systems, due to stochastic traffic demands and limited radio resources, it is typically infeasible to strictly satisfy all QoS constraints at every scheduling instant. Therefore, SLA satisfaction is commonly evaluated from a statistical or long-term perspective, rather than enforced as a hard instantaneous constraint [24].

In this work, SLA satisfaction is assessed based on the quality of service (QoS) compliance ratio within each scheduling window. Specifically, for each slice, its QoS satisfaction level is measured by the fraction of packets whose delays do not exceed a predefined latency threshold, see Section 3.3.3 for details of the reward function.

3.3. DRL-Based Slicing

DRL is a branch of artificial intelligence that focuses on learning optimal decision-making policies through interactions with an environment, rather than relying on labeled data as in supervised learning. An RL agent iteratively observes the environment, takes actions, and receives feedback in the form of rewards, with the objective of maximizing long-term cumulative returns [25].

In network slicing scenarios, resource allocation can be naturally formulated within the RL framework, where the agent learns adaptive slicing decisions based on observed network conditions [26]. Due to the uncertainty in traffic arrivals and the incompleteness of system observations, the considered slicing problem is modeled as a POMDP, in which the agent makes decisions based on partial observations of the underlying system state.

The objective of RL is to learn an optimal policy π^* that maximizes the expected discounted return, given by

$$\pi^* = \arg\max_{\pi} \mathbb{E}_{\pi}\left[\sum_{t=0}^{\infty} \gamma^t R_t\right], \tag{3}$$

where $\gamma \in (0, 1]$ is the discount factor, R_t denotes the reward obtained at scheduling step t, and actions are sampled according to the policy $\pi(\cdot \mid o_t)$ conditioned on the observation o_t.

This formulation enables the DRL agent to learn adaptive slicing policies that account for long-term QoS performance and system stability under dynamic and uncertain traffic conditions.

3.3.1. State Space

The state space is designed to characterize the relative backlog levels across service slices. In this study, we consider three service types: VoLTE, uRLLC, and video services. At each scheduling step, the agent observes the relative proportions of buffered traffic demands associated with these services, which reflect the accumulated unserved service requirements.

Let $b_{\text{volte}}(t)$, $b_{\text{urllc}}(t)$, and $b_{\text{video}}(t)$ denote the buffered traffic demands of the VoLTE, uRLLC, and video slices at the beginning of scheduling step t, respectively. The total buffered demand is defined as

$$b_{\text{total}}(t) = b_{\text{volte}}(t) + b_{\text{urllc}}(t) + b_{\text{video}}(t). \tag{4}$$

When the total buffered demand is non-zero, the relative backlog ratio for each slice is computed as follows.

Relative backlog of VoLTE:

$$\theta_{\text{volte}}(t) = \frac{b_{\text{volte}}(t)}{b_{\text{total}}(t)}, \tag{5a}$$

Relative backlog of uRLLC:

$$\theta_{\text{urllc}}(t) = \frac{b_{\text{urllc}}(t)}{b_{\text{total}}(t)}, \tag{5b}$$

Relative backlog of video:

$$\theta_{\text{video}}(t) = \frac{b_{\text{video}}(t)}{b_{\text{total}}(t)}. \tag{5c}$$

Accordingly, the observation returned to the agent at scheduling step t is defined as

$$\boldsymbol{\theta}(t) = [\theta_{\text{volte}}(t), \theta_{\text{urllc}}(t), \theta_{\text{video}}(t)]. \tag{6}$$

It is worth noting that the state vector only contains relative backlog ratios and does not include information about newly arriving traffic within the current scheduling window. Therefore, the slicing problem is naturally modeled as a POMDP, where traffic arrivals are unobserved yet influence both the system dynamics and reward outcomes.

This state representation allows the agent to make resource allocation decisions based on persistent congestion conditions while avoiding unrealistic assumptions about future traffic, thereby aligning the learning process with practical online RAN slicing scenarios.

3.3.2. Action Space

The action space specifies how the agent allocates radio resources among different service slices. In the considered RAN slicing system, each action corresponds to a resource allocation decision that determines the amount of radio resources assigned to the VoLTE, uRLLC, and video slices.

At each scheduling step, the DRL agent outputs a continuous action vector

$$\mathbf{a}(t) = [a_{\text{VoLTE}}(t),\, a_{\text{uRLLC}}(t),\, a_{\text{Video}}(t)], \tag{7}$$

where $a_i(t) \geq 0$ represents the resource share for slice i. Accordingly, the action space is defined as a continuous space,

$$\mathcal{A} \subset \mathbb{R}^3_{\geq 0}. \tag{8}$$

The total amount of radio resources available per TTI is denoted by

$$R = R_{\max}, \tag{9}$$

which represents the maximum number of PRBs that can be allocated within a single TTI.

To satisfy the system-level resource constraint, the action vector is normalized to obtain a resource allocation weight vector,

$$\tilde{r}_i(t) = \frac{a_i(t)}{\sum_j a_j(t)}, \quad i \in \{\text{VoLTE}, \text{uRLLC}, \text{Video}\}. \tag{10}$$

The normalized weights are then mapped to integer-valued PRBs allocations,

$$\mathbf{r}(t) = [r_{\text{VoLTE}}(t),\, r_{\text{uRLLC}}(t),\, r_{\text{Video}}(t)], \tag{11}$$

which satisfy the following constraint:

$$\sum_i r_i(t) = R, \tag{12}$$

where $r_i(t)$ denotes the number of PRBs allocated to slice i in each TTI.

The selected action is executed once per scheduling step, and the corresponding resource allocation remains fixed across all TTIs within the associated scheduling window.

3.3.3. Reward Function

The reward function plays a critical role in guiding the DRL training process. In the considered RAN slicing scenario, the reward is designed to jointly account for QoS satisfaction and system stability, while respecting heterogeneous SLA priorities across slices.

At each scheduling window t, for each slice i, the QoS satisfaction level is measured by the fraction of packets whose delays do not exceed a predefined latency threshold. Let n_i^{succ} denote the number of packets within the scheduling window that successfully meet the latency requirement, and let n_i^{tot} denote the total number of packets arriving in the same window. The QoS compliance ratio of slice i is defined as

$$\eta_i = \frac{n_i^{\text{succ}}}{n_i^{\text{tot}}}. \tag{13}$$

To capture heterogeneous SLA priorities across slices, we adopt a weighted SLA satisfaction metric. Let $\mathbf{w} = [w_1, w_2, \ldots, w_N]$ denote the SLA weight vector, where $w_i \geq 0$ and $\sum_{i=1}^{N} w_i = 1$. The overall SLA satisfaction level of the system is computed as

$$\mathcal{S} = \sum_{i=1}^{N} w_i \, \eta_i. \tag{14}$$

In addition to delay-based QoS compliance, we introduce penalties for excessive long-term backlog to prevent sustained service degradation. Let q_i denote the resource demand buffered in slice i at the end of a scheduling window, and let r_i denote the amount of resources allocated to slice i per TTI. The normalized backlog ratio is defined as

$$\beta_i = \frac{q_i}{r_i \cdot T_{\text{win}}}, \tag{15}$$

where T_{win} denotes the number of TTIs within a scheduling window.

The backlog penalty for slice i is formulated as a monotonically decreasing function of β_i, reflecting reduced SLA satisfaction under persistent congestion. The overall backlog penalty is computed as a weighted sum across slices using the same SLA weight vector $\mathbf{w}$.

Finally, the system-level SLA objective combines QoS compliance and backlog control into a composite SLA evaluation metric:

$$\mathcal{R}_{\text{SLA}} = \sum_{i=1}^{N} w_i \, \eta_i \; + \; \sum_{i=1}^{N} w_i \, \phi(\beta_i), \tag{16}$$

where $\phi(\cdot)$ denotes the backlog penalty function.

Specifically, the backlog penalty function adopts a piecewise linear form to penalize excessive queue backlog:

$$\phi(\beta_i) = \begin{cases} 0, & \text{if } \beta_i \le \beta_{\min} \\ -\alpha(\beta_i - \beta_{\min}), & \text{if } \beta_{\min} < \beta_i \le \beta_{\max} \\ -\alpha(\beta_{\max} - \beta_{\min}), & \text{if } \beta_i > \beta_{\max} \end{cases} \tag{17}$$

where $\beta_{\min}$ and $\beta_{\max}$ denote the lower and upper tolerance thresholds for backlog, respectively, and α is the penalty coefficient. In our experiments, we set $\beta_{\min} = 0.5$, $\beta_{\max} = 2.0$, and $\alpha = 10$; this formulation enables a flexible trade-off between short-term QoS satisfaction and long-term system stability, making it well suited for RL-based RAN slicing control.

For a given SLA weight configuration, the reward function of the RL agent remains fixed during training, and the learned policy gradually adapts to the specified SLA priorities. Under this setting, the agent implicitly learns a resource allocation strategy that maximizes the weighted QoS compliance.

4. CoMEx: Continual Mixture of Experts Expansion

This section introduces the CoMEx framework. To address the diversity and time-varying nature of SLA requirements in network slicing, CoMEx adopts a hierarchical DRL paradigm consisting of multiple expert networksspecialized for particular SLA configurations and a gating network responsible for dynamic expert scheduling.

We consider a slicing system comprising three representative services, i.e., VoLTE, uRLLC, and Video, whose SLA preference is characterized by a weight vector $\mathbf{w} = [w_1, w_2, w_3]$. The vector is defined on the standard simplex Δ^2. The key novelty of CoMEx is to explicitly incorporate $\mathbf{w}$ as contextual information into the MoE gating policy, thereby enabling conditional adaptation of the policy to different service objectives. CoMEx mainly consists of the following three components:

- Experts Pool: A set of K pre-trained expert policies $\{\pi_k\}_{k=1}^{K}$, where each expert is optimized for a specific subregion of the SLA preference space. After pre-training, the expert parameters are frozen to ensure stable reuse in subsequent stages.
- MoE Gating: A gating policy g_ϕ parameterized by ϕ. The gating network conditions on the environment state and the SLA weight vector to dynamically output expert mixing coefficients.
- Masked Expansion: A binary-mask-based mechanism that dynamically manages the set of available experts. This mechanism enables seamless integration of new experts without changing the gating interface, and supports incremental training to absorb new capabilities, realizing lifelong learning and continual policy evolution.

4.1. Expert Pre-Training and Parameter Freezing

The first step in constructing the expert pool is to select a set of representative SLA weight anchors $\{\mathbf{w}^{(k)}\}_{k=1}^{K} \subset \Delta^2$. Following the principles of coverage and interpretability, these anchors are strategically positioned toward the vertices of the SLA simplex, representing scenarios dominated by specific traffic types (e.g., VoLTE, uRLLC, and Video). This vertex-oriented configuration ensures that each expert develops specialized scheduling capabilities for distinct preference regions while maintaining fundamental coverage of the weight space with a minimal number of experts.

For each anchor $\mathbf{w}^{(k)}$, we train a specialized policy π_k. Upon completion, the parameters of π_k are fully frozen to form a stable "skill library." This freezing strategy is adopted to minimize online retraining costs, prevent the degradation of previously acquired expertise during continual expansion, and ensure system stability by avoiding the risks of online exploration. Consequently, the experts serve solely as fixed inference modules, allowing the subsequent gating network to focus exclusively on high-level expertise composition and rapid adaptation without the interference of coupled parameter updates.

4.2. SLA-Aware Dynamic Gating Mechanism

To make the scheduling policy sensitive to changes in SLA objectives, we construct an augmented observation vector that concatenates the environment observation $\mathbf{o}_t$ with the task weight vector $\mathbf{w}$:

$$\tilde{\mathbf{o}}_t = [\mathbf{o}_t, \mathbf{w}]. \tag{18}$$

At each time step t, the gating network $g_\phi(\tilde{\mathbf{o}}_t)$ outputs a K-dimensional logit vector $\mathbf{z}_t$. The mixing coefficients $\boldsymbol{\pi}_t$ are computed:

$$\boldsymbol{\pi}_t = \mathrm{softmax}(\mathbf{z}_t/\tau), \quad \text{s.t.} \sum_{k=1}^{K} \pi_{t,k} = 1,\ \pi_{t,k} \geq 0. \tag{19}$$

All frozen experts then perform parallel inference under the current observation $\mathbf{o}_t$ and output candidate actions $\mathbf{a}_t^{(k)}$. The final executed action is obtained by the weighted fusion of these candidate actions:

$$\mathbf{a}_t = \sum_{k=1}^{K} \pi_{t,k}\, \mathbf{a}_t^{(k)}. \tag{20}$$

This mechanism allows the gating network to adapt $\boldsymbol{\pi}_t$ within an episode based on real-time queue states, thereby substantially improving robustness under previously unseen SLA configurations.

The training objective of the gating network is to intelligently dispatch experts according to the current SLA task distribution. We employ Proximal Policy Optimization (PPO) to train the gating network. Through end-to-end RL optimization, the gating network can naturally learn collaborative utilization of expert capabilities, avoiding the out-of-distribution generalization degradation commonly observed in supervised gating, as shown in Algorithm 1.

Algorithm 1 Gating training

```
 1: **Input:** training weight set/distribution $\mathcal{W}_{\text{train}}$, PPO hyperparameters
 2: Initialize gating policy $g_\phi$ and (optional) mask $\mathbf{m}$
 3: **while** not converged **do**
 4:     Sample $\mathbf{w} \sim \mathcal{W}_{\text{train}}$; reset environment with $\mathbf{w}$
 5:     **for** each step $t$ **do**
 6:         Observe $\mathbf{o}_t$; form $\tilde{\mathbf{o}}_t = [\mathbf{o}_t, \mathbf{w}]$
 7:         Compute $\boldsymbol{\pi}_t$ via (19) and mask constraint (21)
 8:         Query frozen experts $\{\pi_k\}$ to obtain $\{\mathbf{a}_t^{(k)}\}$; fuse action by (22)
 9:         Execute $\mathbf{a}_t$, receive $r_t, \mathbf{o}_{t+1}$; store transition
10:     **end for**
11:     Update $\phi$ by PPO
12: **end while**
```

4.3. Masked Continual Expansion

Relying on a fixed number of experts limits the upper bound of the system's representational capacity. When the SLA objective drifts or new service demands emerge, CoMEx introduces a continual expansion mechanism to overcome performance bottlenecks.

This mechanism dynamically controls expert availability via a binary mask vector $\mathbf{m} \in \{0,1\}^{K_{\max}}$. Let the currently activated expert set be $\mathcal{K} = \{k \mid m_k = 1\}$. The gating output is then constrained to lie within the activated set:

$$\pi_{t,k} = 0, \ \forall k \notin \mathcal{K}, \qquad \sum_{k \in \mathcal{K}} \pi_{t,k} = 1. \tag{21}$$

Accordingly, the action fusion rule is modified to aggregate only over the activated experts:

$$\mathbf{a}_t = \sum_{k \in \mathcal{K}} \pi_{t,k}\, \mathbf{a}_t^{(k)}. \tag{22}$$

When additional capacity is required, the system trains a new expert π_{K+1}, freezes it, and appends it to the expert pool, followed by updating the mask $\mathbf{m}$. Subsequently, only incremental training of the gating network is needed. This design preserves the capabilities of existing experts, mitigates the risk of catastrophic forgetting at the mechanism level, while enabling flexible continual evolution of the overall system, as summarized in Algorithm 2.

Algorithm 2 Expert expansion and inference

```
 1: **Input:** expert pool $\{\pi_k\}_{k=1}^{K}$, gating $g_\phi$, mask $\mathbf{m}$
 2: **if** need expansion **then**
 3:     Train new expert $\pi_{K+1}$ under $\mathbf{w}^{(K+1)}$; freeze parameters and add to pool
 4:     Update mask: $\mathbf{m}_{K+1} \leftarrow 1$; fine-tune $g_\phi$ via PPO
 5: **end if**
 6: **Inference:** given test SLA weight $\mathbf{w}$
 7: **for** each step $t$ **do**
 8:     Form $\tilde{\mathbf{o}}_t = [\mathbf{o}_t, \mathbf{w}]$; compute $\boldsymbol{\pi}_t = g_\phi(\tilde{\mathbf{o}}_t)$
 9:     Query active experts and fuse: $\mathbf{a}_t = \sum_{k:\mathbf{m}_k=1} \pi_{t,k}\mathbf{a}_t^{(k)}$
10:     Execute $\mathbf{a}_t$
11: **end for**
12: **Output:** adapted policy trajectory
```

4.4. Complexity Analysis and Implementation Overhead

4.4.1. Computational Complexity and Inference Latency

At each decision step t, the overall inference latency T_{total} consists of three stages: gating decision, parallel expert inference, and action fusion. Let the gating network g_ϕ be a multi-layer perceptron (MLP) with L_g layers and D_g hidden units per layer, whose time complexity is $\mathcal{O}(L_g D_g^2)$. Since the K expert policies $\{\pi_k\}_{k=1}^K$ are architecturally independent, on computing platforms that support GPU parallelism or multi-core CPU scheduling, the computation of expert action generation can be treated as effectively constant, i.e., $T_{\text{exp}} \approx \max_k\{T(\pi_k)\}$. The action fusion stage follows (20) and has complexity $\mathcal{O}(K \cdot d_{\text{act}})$, where d_{act} is the dimensionality of the action vector. Therefore, the per-step total time complexity of CoMEx can be expressed as

$$T_{\text{total}} = \mathcal{O}(L_g D_g^2) + \max_{k=1,\dots,K}\left\{\mathcal{O}(L_\pi D_\pi^2)\right\} + \mathcal{O}(K \cdot d_{\text{act}}). \tag{23}$$

Because the gating network is typically designed to be lightweight ($D_g \ll D_\pi$) and the expert networks are of moderate size, the theoretical analysis in this section suggests that the inference latency of CoMEx can satisfy the stringent delay constraints of 5G/6G scheduling cycles, which are typically on the millisecond scale. The above statement should be interpreted as a theoretical implication rather than an empirical timing result.

4.4.2. Space Complexity and Storage Overhead

The storage requirement of the system is mainly determined by the parameter scale. Since the expert parameters are frozen after training, the total number of stored parameters P_{total} grows linearly with the number of experts K:

$$P_{\text{total}} = P_\phi + \sum_{k=1}^{K} P_{\pi,k} \approx P_\phi + K \cdot \bar{P}_\pi, \tag{24}$$

where $\bar{P}_\pi$ denotes the average parameter count of a single expert. In the continual expansion setting, the masked expansion mechanism reserves a maximum expert capacity $K_{\max}$. Since the mixing weight vector $\boldsymbol{\pi}_t$ requires only a few bytes to store, this mechanism is computationally efficient under constrained storage budgets.

4.4.3. Training Efficiency Analysis

The training procedure of CoMEx is decomposed into two decoupled stages, which significantly improves resource utilization:

- Parallel pre-training stage: The K experts can be trained in parallel on different SLA anchor tasks without interference, effectively reducing the initial convergence time.
- Incremental optimization stage: When a new expert is introduced, the existing experts remain frozen; thus, the system only needs to update the relatively small gating network parameters ϕ using PPO. This incremental learning paradigm avoids the substantial computational cost of retraining from scratch and mitigates the risk of catastrophic forgetting at the mechanism level, a common issue in RL.

Overall, CoMEx achieves a favorable balance between computational efficiency and flexibility, providing robust technical support for slice resource allocation under dynamic SLA environments.

5. Experiments

5.1. Experimental Setup

The programs used in this study were run on a high-performance AI server equipped with dual Intel Xeon Silver 4309Y CPUs @ 2.80 GHz and NVIDIA RTX 4090 GPU, with 128 GB of RAM, running Ubuntu 22.04. All simulation experiments were implemented in Python 3.8.18.

Our experiments are conducted on an open-source RAN slicing reinforcement learning simulation environment publicly available on GitHub (https://github.com/ahmadnagib/TL4RL, accessed on 12 January 2026). The environment follows a standard Gym-style interface, facilitating interoperability with mainstream RL libraries. Building upon the original implementation, we further extend and calibrate the simulator to better match the needs of this study, enabling more faithful modeling of time-varying wireless channel characteristics, heterogeneous traffic processes, and practical resource constraints.

In this simulation setup, we instantiate three representative network slices, namely VoLTE, uRLLC, and video, each with distinct traffic patterns and QoS requirements. The packet inter-arrival processes and packet-size distributions of different slices are modeled to reflect realistic RAN characteristics.

Specifically, VoLTE traffic is modeled with a uniform inter-arrival process and fixed packet size, representing quasi-periodic and delay-sensitive voice services. uRLLC traffic exhibits strong burstiness and stringent latency requirements; thus, its packet arrival times and packet sizes are generated using truncated heavy-tailed distributions. Video traffic is modeled as bandwidth-intensive and relatively delay-tolerant, where packet inter-arrival times follow an exponential distribution and packet sizes are drawn from a truncated log-normal distribution. The detailed configurations are summarized in Table 1.

Table 1. Traffic generation and scheduling configuration for RAN services.

	VoLTE	**uRLLC**	**Video**
Scheduling mechanism	Window-based allocation with TTI-level Round Robin		
Scheduling window size	40 transmission time intervals (TTIs)		
Packet inter-arrival time distribution	Uniform (Min = 0, Max = 160 ms)	Truncated heavy-tailed distribution (Mean = 6 ms, Max = 12.5 ms)	Exponential (Mean = 180 ms)
Packet size distribution	Constant (40 Byte)	Truncated heavy-tailed distribution (Mean = 100 Byte, Max = 250 Byte)	Truncated log-normal distribution (Mean = 2 MB, Std = 0.722 MB, Max = 5 MB)

Different slice services exhibit heterogeneous latency sensitivities. According to 3GPP specifications, we define tiered QoS objectives for the three slice types: the VoLTE slice targets an average packet delay no greater than 30 ms, the uRLLC slice targets no greater than 10 ms, and the video slice targets no greater than 80 ms, see Table 2. The reward function is designed as a delay-based linear scoring mechanism: for each slice, a QoS score is computed by linear interpolation between the target delay and an upper tolerance bound, and the overall reward is obtained via a weighted sum using the service weights. This design enables the agent to achieve resource-efficient allocation under dynamic SLA weight configurations while respecting differentiated latency constraints. Compared with stepwise rewards, linear interpolation avoids vanishing-gradient issues and provides a smoother learning signal.

Table 2. Delay tolerance for different slices.

Slice Type	Target Delay (ms)	Delay Tolerance (ms)
VoLTE	20	10
uRLLC	5	5
Video	50	30

In the simulation environment, to balance decision stability with the real-time nature of underlying transmissions, we design a two-level scheduling framework: (1) Window-level resource allocation: the agent makes one resource allocation decision at the beginning of each scheduling window and outputs a normalized three-dimensional action vector, representing the resource proportions assigned to the three slices within that window. The decision remains fixed throughout the window, avoiding excessive policy fluctuations that may induce system oscillations, and is consistent with practical 5G control-plane update periodicities. (2) TTI-level packet scheduling: at each TTI, given the window-level resource budget, a Round-Robin mechanism is employed to serve active flows across slices. Each training episode contains 100 scheduling windows, which is sufficient to capture traffic burstiness and observe policy convergence behavior. The complete simulation parameters are summarized in Table 3. This design both preserves the learnability of RL decisions and meets the millisecond-level scheduling responsiveness required by real-time services.

Table 3. Simulation environment parameters.

Parameter	Value	Definition
max_episode_timesteps	100	Number of timesteps per episode.
sl_win_size	40	Scheduling window size: 40 TTI = 40 ms.
time_quantum	1	Time quantum: 1 TTI = 1 ms.
max_trans_per_tti	120	Maximum number of transmissions per TTI.
max_size_per_tti	85	Maximum resource capacity per TTI.

5.2. Experimental Process

In this experiment, we first pre-train three expert policies using RL, where each expert is specialized for a particular service type. Specifically, we construct three experts with service-dominant SLA weight configurations, so that each expert can achieve strong performance on resource scheduling for its targeted service. The expert configurations are summarized as follows:

- VoLTE expert: the weight configuration is $[0.50, 0.30, 0.20]$, where VoLTE has the highest weight, making the expert dominant in VoLTE-oriented scheduling.
- uRLLC expert: the weight configuration is $[0.20, 0.50, 0.30]$, where uRLLC has the highest weight, ensuring prioritized resource allocation for services with stringent latency and reliability requirements.
- Video expert: the weight configuration is $[0.30, 0.20, 0.50]$, where video has the largest weight, making it suitable for managing bandwidth-intensive video traffic.

We train all experts using PPO. Each expert is trained on the RAN slicing environment described above, and its policy is optimized to maximize the weighted long-term return under the corresponding SLA preference. The training of experts is conducted independently; by adjusting the slice weight vector during training, each expert is encouraged to focus on its targeted service type and to learn a specialized scheduling behavior.

The PPO training hyperparameters are reported in Table 4. With this pre-training procedure, each expert is ensured to achieve strong performance within its specialized service domain, thereby providing a solid foundation for the subsequent incremental learning and expert fusion stages.

Table 4. PPO algorithm hyperparameters.

Parameter	Value
Learning rate	1×10^{-4}
Entropy coefficient	0.05
Batch size	512
Number of steps	1024
Discount factor γ	0.99
GAE parameter λ	0.95
Clip range ϵ	0.2

Based on the frozen expert policies, we construct a MoE environment for training the gating network. This environment is built upon the underlying slicing simulator and takes a target SLA weight vector $\mathbf{w}$ as input to instantiate the corresponding base environment, thereby supporting training and evaluation under different SLA configurations. To make the gating policy explicitly aware of the SLA objective, the environment concatenates the base observation $\mathbf{o}(t)$ with the current target weight $\mathbf{w}$, forming an augmented observation that is fed into the gating network. Meanwhile, the action space of the gating policy is defined as a continuous logits vector over the expert dimension. In the continual expansion setting, the action dimensionality can be extended and, together with a masking mechanism, the interface can remain consistent while enabling incremental learning. Within this gating-training environment, we also adopt PPO to optimize the gating policy, so that it can adaptively produce expert mixing coefficients under varying SLA weights, effectively reusing existing experts and achieving fast policy adaptation to unseen SLA configurations.

To further improve adaptability over a broader range of SLA preferences, we select a new expert weight vector $\mathbf{w}_4 = [0.3, 0.3, 0.4]$ as the training target for an additional expert. This weight lies in the interior of the standard simplex Δ^2, providing a relatively balanced service prioritization and thus serving as a complement to the three existing experts.

During training, we train the new expert using the same PPO hyperparameters as those used for the first three experts to ensure fairness and reproducibility. After training, the new expert policy is likewise frozen and added to the expert pool for subsequent incremental training of the gating network.

Finally, without retraining any existing experts, we continue to incrementally train the gating network within the original MoE framework, enabling it to learn improved fusion strategies over the expanded expert set. Accordingly, the gating output is extended from three dimensions to four dimensions. Through this procedure, CoMEx can achieve faster and more stable policy adaptation under SLA shifts and previously unseen weight configurations.

5.3. Experimental Results

To mitigate the impact of randomness and ensure statistical rigor, all experiments are repeated 20 times and evaluated using the average performance metrics. First, for the three pre-trained expert policies, we test each expert in the training environment to verify its specialization and generalization under different SLA configurations. Among the probe weights, [0.33, 0.34, 0.33] corresponds to an approximately balanced SLA scenario, whereas the other three weight settings represent VoLTE-, uRLLC-, and video-dominant scenarios, respectively. The results show that under the balanced weight configuration, the three experts achieve comparable average returns, indicating that each expert retains a certain degree of general scheduling capability. In contrast, under the other three scenarios, clear specialization differentiation emerges: each expert attains higher returns when evaluated on SLA configurations that match or are close to its pre-training weight preference. This observation validates both the complementarity of the expert pool and its division-of-labor property. The above phenomenon is visually illustrated in Figure 4.

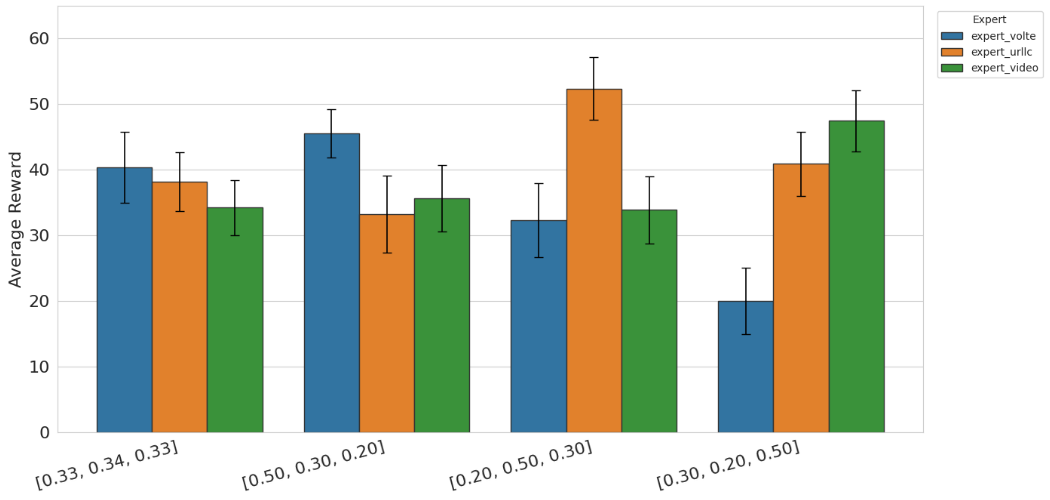

Figure 4. Performance comparison of the three pre-trained experts evaluated on the standard probe SLA weights. Error bars denote the standard deviation.

To systematically evaluate the generalization capability of the gating policy under unseen SLA configurations, we construct a test weight set $\mathcal{W}_{\text{test}}$ over the standard simplex Δ^2, as shown in Table 5. The design of $\mathcal{W}_{\text{test}}$ follows three principles: (1) Coverage: the test points should span different regions of the simplex, including extreme preference regions near vertices, two-service-dominant regions near edges, and balanced regions near the center; (2) Out-of-distribution nature: the test points should be sufficiently different from the experts' pre-training weights to assess the gate's adaptation capability under preference shifts; and (3) Representativeness: the test points should reflect typical weight patterns that may arise in real operations, thereby matching practical SLA adjustment behaviors.

Regarding out-of-distribution generalization to extreme or dynamically changing weights, our evaluation focuses on unseen preferences within the standard simplex because SLA weights in real RAN slicing are priority proportions and thus naturally satisfy simplex constraints. We also avoid fully degenerate extremes (e.g., exactly one-hot weights) as they are uncommon in operation and can trivially starve other services. While the chosen test points include near-vertex configurations to stress the preference shift, they remain feasible under the simplex formulation.

Table 5. Selected test SLA weights $\mathcal{W}_{\text{test}}$ for evaluating generalization.

ID	Test Weight $\mathbf{w} \in \Delta^2$
T1	[0.80, 0.10, 0.10]
T2	[0.50, 0.10, 0.40]
T3	[0.55, 0.25, 0.20]
T4	[0.22, 0.19, 0.59]
T5	[0.40, 0.50, 0.10]
T6	[0.17, 0.60, 0.23]
T7	[0.33, 0.34, 0.33]
T8	[0.11, 0.41, 0.48]

On the above eight unseen test SLA weight configurations, we systematically evaluate the generalization performance of the three standalone expert policies and the CoMEx gating policy. As shown in Figure 5, the error bars represent the standard deviation over 20 independent runs. the results can be summarized as follows:

Limitations of single-expert policies: The three experts (`expert_volte`, `expert_urllc`, and `expert_video`) perform well under their preferred SLA configurations, yet their performance degrades sharply under mismatched objectives. For example, under the VoLTE-dominant setting [0.8, 0.1, 0.1], `expert_volte` achieves a reward of 61.12, whereas `expert_urllc` attains only 17.73, corresponding to a 244.8% gap. Similarly, under the uRLLC-dominant setting [0.17, 0.6, 0.23], `expert_urllc` (55.38) significantly outperforms `expert_volte` (31.84). This pronounced objective dependence confirms that a single policy is fundamentally insufficient for dynamic SLA requirements.

Adaptability of the gating policy: In contrast, the CoMEx gating network (`RL_gating`) achieves consistently high performance across all eight test configurations, with rewards ranging from 77.54 to 80.23. It attains an average reward of 78.95 with a standard deviation of only 0.88. Notably,

- Under the most challenging near-balanced setting [0.33, 0.34, 0.33], the gating policy (78.95) improves over the best single expert (`expert_urllc`: 41.10) by 92.1%;
- Under the extreme setting [0.4, 0.5, 0.1], the gating policy (80.23) improves over the best single expert (`expert_volte`: 45.98) by 74.5%;
- The per-configuration reward variability of the gating policy ($1.76 \sim 2.99$) is substantially lower than that of single experts ($3.31 \sim 8.67$), indicating superior robustness and stability.

Consistency across the weight space: By comparing the bar heights under different SLA configurations in Figure 5, we observe that single experts exhibit large fluctuations (e.g., `expert_urllc` drops to 17.73 at [0.8, 0.1, 0.1] but rises to 55.38 at [0.17, 0.6, 0.23]), whereas the gating policy maintains high and smooth performance throughout. These results indicate that CoMEx successfully learns to adjust expert mixing coefficients conditioned on the SLA objective, effectively leveraging expert complementarity to achieve fast zero-shot adaptation under unseen SLA configurations.

To further validate the performance gain and stability of CoMEx under continual expansion, we compare the three-expert RL gating configuration (`Gating_RL`) with the four-expert expanded configuration (CoMEx) on the eight unseen SLA weight vectors in $\mathcal{W}_{\text{test}}$. Figure 6 reports the average reward achieved at each test weight, with the shaded bands indicating the variability (± 1 standard deviation) over 20 independent runs.

Substantial performance improvement: As shown in Figure 6, the four-expert CoMEx configuration consistently achieves higher average rewards than the three-expert RL gating baseline across the eight unseen test SLA weight configurations, indicating that continual expansion can improve performance under diverse unseen SLA objectives.

Stability under unseen weights: The shaded bands in Figure 6 visualize the variability across runs for each test weight. CoMEx maintains comparable variability to the three-expert baseline while delivering higher rewards across all test points, suggesting that the observed improvement is not accompanied by increased instability.

Benefit of expansion with a new expert: The performance gain is obtained after adding an interior expert and updating the gating policy accordingly, while keeping the existing experts unchanged. This supports the practical value of CoMEx as a scalable approach for extending expert coverage under long-term SLA drift.

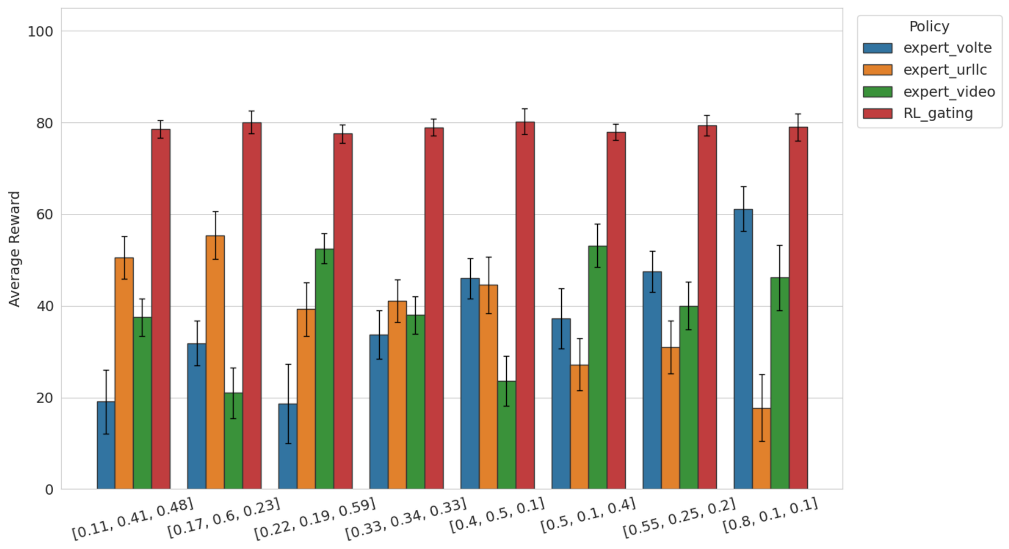

Figure 5. Performance and stability analysis under unseen SLA weight configurations. Error bars denote the standard deviation.

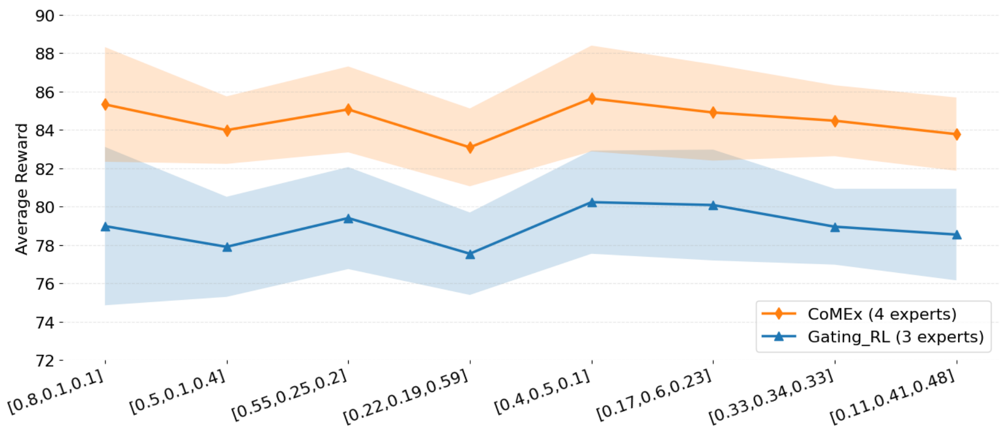

Figure 6. Performance comparison of Gating_RL and CoMEx over the eight unseen test SLA weight configurations. Markers denote the mean average reward for each weight, and the shaded bands indicate ±1 standard deviation.

5.4. Ablation Study

To further examine the effectiveness of key design choices in CoMEx, we conduct ablation studies along two dimensions: (1) supervised gating versus reinforcement-learning-based gating; and (2) episode-level versus step-level gating granularity. Table 6 reports a detailed comparison across the eight unseen test SLA weight configurations.

Table 6. Performance comparison of different gating training strategies.

Test Weight	SP	RL-Episode	RL-Step	RL-Inc	Best Gain
$[0.80, 0.10, 0.10]$	61.89	76.23	78.98	85.33	+37.8%
$[0.50, 0.10, 0.40]$	52.86	76.52	77.90	83.99	+58.9%
$[0.55, 0.25, 0.20]$	52.14	76.84	79.40	85.07	+63.2%
$[0.22, 0.19, 0.59]$	50.45	76.99	77.54	83.09	+64.7%
$[0.40, 0.50, 0.10]$	76.62	77.47	80.23	85.64	+11.8%
$[0.17, 0.60, 0.23]$	53.59	77.94	80.08	84.91	+58.5%
$[0.33, 0.34, 0.33]$	68.89	77.21	78.95	84.48	+22.6%
$[0.11, 0.41, 0.48]$	78.25	77.61	78.54	83.78	+7.1%
Average	61.84	77.10	78.95	84.54	+36.7%

5.4.1. Supervised Gating vs. RL Gating

The first ablation study compares supervised gating (SP) with rl-based gating (RL-episode). The SP gate is trained in a supervised manner as an offline soft-label gating mapping baseline. Specifically, for each training SLA weight vector $\mathbf{w}$, we independently and offline-evaluate every frozen expert in the pool under the same preference $\mathbf{w}$, obtaining an average return (equivalently, a weighted QoS score) for each expert. We then apply a Softmax normalization over these expert scores to construct a soft supervision target, which serves as the optimal fusion-coefficient label under $\mathbf{w}$. The gating network is subsequently trained to regress a mapping from $\mathbf{w}$ to the fusion weights.

As reported in Table 6, SP achieves an average reward of 61.84 over the eight unseen test configurations, whereas RL-episode attains 77.10, yielding a 24.7% improvement. It is worth noting that although SP is substantially worse than RL-episode on average, its performance is comparable to RL-episode at certain probe weights (e.g., $[0.4, 0.5, 0.1]$ and $[0.11, 0.41, 0.48]$), indicating that SP is not uniformly ineffective but is more sensitive to the choice of SLA weights.

Analysis: The limitations of supervised gating can be attributed to the following factors:

- Suboptimality of offline evaluation: The supervision targets in SP are derived from offline expert evaluations, whereas the experts' performance during online interaction can deviate due to distribution shift. In particular, when the test weights are far from the training distribution, the reliability of offline evaluation deteriorates, leading to inaccurate supervision signals.
- Lack of end-to-end optimization: SP decouples "performance estimation" from "gate learning" and thus cannot optimize the fusion policy end-to-end. In contrast, RL-episode is trained directly under the target reward, enabling it to learn fusion strategies that better reflect the underlying dynamics.
- Limited generalization: Supervised learning depends on the coverage of the training samples. When a test weight lies in a sparse region of the training distribution, SP receives insufficient supervision and may therefore fail to generalize.

These results highlight the necessity of rl-based gating: by learning from online interaction, the RL gate can discover more robust expert fusion patterns in a dynamic environment, leading to substantially improved zero-shot generalization.

5.4.2. Episode-Level vs. Step-Level Gating Decisions

In MoE architecture, the temporal granularity at which the gating network outputs fusion coefficients is a key factor affecting policy performance. The second ablation study investigates this effect by comparing episode-level and step-level gating. RL-episode determines the expert fusion coefficients π_t at the beginning of each episode based on the target SLA weight vector $\mathbf{w}$, and keeps π_t fixed throughout the episode. In contrast, RL-step updates the fusion coefficients at every scheduling window (i.e., step) to adapt to the evolving system dynamics. As shown in Table 6, RL-step achieves an average reward of 78.95, improving over RL-episode (77.10) by 2.4%, with a 95% CI of [1.15, 2.55], $p = 0.0004$ ($p < 0.001$). This confirms the substantial advantage of step-level decision granularity over episode-level gating.

Analysis: The performance advantage of step-level gating stems from:

- Adaptation to time-varying traffic: Within an episode, the traffic demand (reflected in $\mathbf{o}(t)$) can change substantially over time due to burstiness and periodic patterns. RL-step can adjust expert weights based on the real-time state, enabling finer-grained adaptation.

- Dynamic trade-off between exploitation and correction: Episode-level gating enforces a fixed fusion strategy over the entire episode, which may become suboptimal at certain steps. Step-level gating allows the policy to revise the expert mixture online as the state evolves, thereby mitigating persistent suboptimal dispatch decisions.
- Increased expressiveness: Step-level gating operates in a richer policy class (time-varying fusion versus static fusion), which can capture more complex conditional dependencies between network states and expert utilization.

Despite the superior performance of RL-step, episode-level gating has practical advantages for deployment: (1) lower computational overhead; (2) potentially improved operational stability by avoiding frequent expert switching and the resulting resource oscillations; and (3) better interpretability, since the fusion coefficients depend only on $\mathbf{w}$, which facilitates operator understanding and intervention. Therefore, when compute budgets or stability constraints are stringent, episode-level gating can provide a favorable trade-off between performance and efficiency.

5.4.3. Benefits of Continual Expansion

As an additional validation, Table 6 also reports the performance of four different gating training paradigms. Building upon RL-step, RL-inc introduces a fourth balanced expert and performs incremental training of the gating network. RL-inc achieves an average reward of 84.54, improving over RL-step by 7.1% and over the supervised baseline (SP) by 36.7%. These results confirm the effectiveness of CoMEx's continual expansion mechanism and provide a practical pathway for continual system evolution under drifting SLA objectives.

Overall, the ablation study highlights the key design choices in CoMEx: (1) RL-based gating yields a 24.7% improvement over supervised gating and is essential for robust zero-shot generalization; (2) step-level dispatching improves performance by 2.4% over episode-level gating while allowing flexible trade-offs between performance and efficiency; and (3) the continual expansion mechanism provides an additional 7.1% gain, validating CoMEx's incremental learning capability. Collectively, these findings offer systematic guidance for adaptive resource management in RAN slicing under dynamic SLAs.

6. Limitations

Although CoMEx demonstrates strong generalization and scalability in experiments, there are several limitations that need to be addressed in future work.

(1) Key engineering constraints for real deployment. In online scheduling scenarios, the decision-making pipeline must meet stringent millisecond-level latency requirements. Real network deployment involves control and user plane interface constraints, observability and data feedback loops, failure recovery strategies, gray release, and version management requirements. Ensuring the stability and auditability of slices while integrating gating updates, expert expansion, and online monitoring feedback loops into the operational maintenance system of live networks is a core challenge that needs to be solved for the engineering implementation of CoMEx.

(2) Expert expansion remains heuristic and lacks systematic triggering criteria. This paper validates the feasibility of continual expansion by adding a "balanced" interior expert and incrementally fine-tuning the gating network; however, when to trigger expansion remains a key practical challenge. Toward deployment, we plan to incorporate two complementary triggers: performance-based triggering (expand when the overall return on a representative SLA distribution noticeably degrades relative to the best single expert) and coverage-based triggering (expand when the online workload frequently encounters SLA configurations far from all existing anchors). Together,

these triggers can enable CoMEx to dynamically grow the expert pool and better cope with diverse and continually evolving SLA requirements in real-world RANs.

(3) Insufficient precise measurement of catastrophic forgetting. The design of "freezing existing expert parameters + masked expansion + incremental training of the gating network" reduces the risk of forgetting from a mechanism perspective. However, current evidence mainly comes from the rationale behind the architecture design and training paradigm, and does not provide direct experimental measurements to quantify the performance retention of old tasks/old weight configurations before and after expansion. Therefore, the conclusion regarding the mitigation of catastrophic forgetting still needs to be verified in future work through more rigorous before-and-after comparison experiments and continuous learning baseline comparisons.

7. Conclusions

This paper tackles policy mismatch and costly retraining in RAN slicing by proposing CoMEx, a continually extensible MoE framework. By combining a frozen expert pool with an RL-based, SLA-aware gating mechanism, CoMEx enables adaptive, step-level fusion of expert actions. Moreover, to cope with long-term SLA drift, it introduces a masked expansion scheme: while preserving existing knowledge, the system can absorb newly added expert capabilities through incremental updates to the gating network only. The proposed method exhibits stronger transferability and robustness under unseen SLA weights. Overall, CoMEx provides a scalable solution for slice policy management in dynamic SLA environments. Future research will focus on the key engineering constraints of deploying CoMEx in real RAN systems, the systematic design and triggering strategies for expert expansion, and more precise quantitative evaluation and validation of catastrophic forgetting.

Author Contributions: X.M.: Conceptualization, Methodology, Writing—Original Draft Preparation. M.L.: Investigation. Y.X.: Validation. D.L.: Supervision and Reviewing. All authors have read and agreed to the published version of the manuscript.

Funding: This work was supported by the Science and Technology Research Project of Jiangxi Provincial Department of Education under Grant GJJ2402602.

Data Availability Statement: The data are available from the corresponding author on reasonable request.

Conflicts of Interest: The authors declare no conflicts of interest.

Abbreviations

The following abbreviations are used in this manuscript:

CoMEx	Continual Mixture of Experts
DRL	Deep Reinforcement Learning
IQR	Interquartile Range
MLP	Multi-layer Perceptron
MNOs	Mobile Network Operators
POMDP	Partially-Observable Markov Decision Process
PPO	Proximal Policy Optimization
PRBs	Physical Resource Blocks
QoE	Quality of Experience
QoS	Quality of Service
RL	Reinforcement Learning
SLA	Service Level Agreement
TTI	transmission time intervals

uRLLC	ultra-Reliable Low-Latency Communications
VoLTE	LTE Voice

References

1. Alam, K.; Habibi, M.A.; Tammen, M.; Krummacker, D.; Saad, W.; Di Renzo, M.; Melodia, T.; Costa-Pérez, X.; Debbah, M.; Dutta, A.; et al. A Comprehensive Tutorial and Survey of O-RAN: Exploring Slicing-Aware Architecture, Deployment Options, Use Cases, and Challenges. *IEEE Commun. Surv. Tutor.* **2025**, *28*, 1637–1678 .
2. Filali, A.; Mlika, Z.; Cherkaoui, S. Open RAN Slicing for MVNOs with Deep Reinforcement Learning. *IEEE Internet Things J.* **2024**, *11*, 18711–18725. [CrossRef]
3. Muntaha, S.T.; Hafeez, M.; Ahmed, Q.Z.; Khan, F.A.; Zaharis, Z.D.; Lazaridis, P.I. RAN Slicing with Joint Resource Allocation for a Multi-Tenant Multi-Service System. *IEEE Trans. Cogn. Commun. Netw.* **2024**, *11*, 1927–1939.
4. Rafique, W.; Barai, J.R.; Fapojuwo, A.O.; Krishnamurthy, D. A Survey on Beyond 5G Network Slicing for Smart Cities Applications. *IEEE Commun. Surv. Tutor.* **2024**, *27*, 595–628. [CrossRef]
5. Nagib, A.M.; Abou-Zeid, H.; Hassanein, H.S. Accelerating Reinforcement Learning via Predictive Policy Transfer in 6G RAN Slicing. *IEEE Trans. Netw. Serv. Manag.* **2023**, *20*, 1170–1183. [CrossRef]
6. Nagib, A.M.; Abou-Zeid, H.; Hassanein, H.S. Safe and Accelerated Deep Reinforcement Learning-Based O-RAN Slicing: A Hybrid Transfer Learning Approach. *IEEE J. Sel. Areas Commun.* **2023**, *42*, 310–325.
7. Gudepu, V.; Chintapalli, V.R.; Castoldi, P.; Valcarenghi, L.; Tamma, B.R.; Kondepu, K. Adaptive Retraining of AI/ML Model for Beyond 5G Networks: A Predictive Approach. In Proceedings of the 2023 IEEE 9th International Conference on Network Softwarization (NetSoft), Madrid, Spain, 19–23 June 2023; pp. 282–286.
8. Chen, Y.H. An Adaptive Heuristic Algorithm to Solve the Network Slicing Resource Management Problem. *Int. J. Commun. Syst.* **2023**, *36*, e5463. [CrossRef]
9. Helmy, M.; Abdellatif, A.A.; Mhaisen, N.; Mohamed, A.; Erbad, A. Slicing for AI: An Online Learning Framework for Network Slicing Supporting AI Services. *IEEE Trans. Netw. Serv. Manag.* **2025**, *22*, 5239–5254. [CrossRef]
10. Wang, H.; Wu, Y.; Min, G.; Xu, J.; Tang, P. Data-Driven Dynamic Resource Scheduling for Network Slicing: A Deep Reinforcement Learning Approach. *Inf. Sci.* **2019**, *498*, 106–116. [CrossRef]
11. Yan, M.; Feng, G.; Zhou, J.; Sun, Y.; Liang, Y.-C. Intelligent Resource Scheduling for 5G Radio Access Network Slicing. *IEEE Trans. Veh. Technol.* **2019**, *68*, 7691–7703. [CrossRef]
12. Nagib, A.M.; Abou-Zeid, H.; Hassanein, H.S. Transfer Learning-Based Accelerated Deep Reinforcement Learning for 5G RAN Slicing. In *Proceedings of the 2021 IEEE 46th Conference on Local Computer Networks (LCN), Virtual, 4–7 October 2021*; IEEE: New York, NY, USA, 2021; pp. 249–256.
13. Zangooei, M.; Golkarifard, M.; Rouili, M.; Saha, N.; Boutaba, R. Flexible RAN Slicing in Open RAN with Constrained Multi-Agent Reinforcement Learning. *IEEE J. Sel. Areas Commun.* **2023**, *42*, 280–294. [CrossRef]
14. Alcaraz, J.J.; Losilla, F.; Zanella, A.; Zorzi, M. Model-Based Reinforcement Learning with Kernels for Resource Allocation in RAN Slices. *IEEE Trans. Wirel. Commun.* **2022**, *22*, 486–501. [CrossRef]
15. Li, J.; Su, Q.; Yang, Y.; Jiang, Y.; Wang, C.; Xu, H. Adaptive Gating in Mixture-of-Experts Based Language Models. *arXiv* **2023**, arXiv:2310.07188.
16. Kong, Y.; Ma, G.; Zhao, Q.; Wang, H.; Shen, L.; Wang, X.; Tao, D. Mastering Massive Multi-Task Reinforcement Learning via Mixture-of-Expert Decision Transformer. In Proceedings of the ICLR 2025 Workshop on Modularity for Collaborative, Decentralized, and Continual Deep Learning, Singapore, 27 April 2025.
17. Li, H.; Lin, S.; Duan, L.; Liang, Y.; Shroff, N.B. Theory on Mixture-of-Experts in Continual Learning. *arXiv* **2024**, arXiv:2406.16437.
18. Ma, L.; Cheng, N.; Zhou, C.; Wang, X.; Lu, N.; Zhang, N. Dynamic Neural Network-Based Resource Management for Mobile Edge Computing in 6G Networks. *IEEE Trans. Cogn. Commun. Netw.* **2023**, *10*, 953–967. [CrossRef]
19. Nagib, A.M.; Abou-Zeid, H.; Hassanein, H.S. Toward Safe and Accelerated Deep Reinforcement Learning for Next-Generation Wireless Networks. *IEEE Netw.* **2022**, *37*, 182–189. [CrossRef]
20. Mu, X.; Xu, Y.; Li, D.; Liu, M. TADocs: Teacher–Assistant Distillation for Improved Policy Transfer in 6G RAN Slicing. *Mathematics* **2024**, *12*, 2934. [CrossRef]
21. Xu, M.; Shen, Y.; Zhang, S.; Lu, Y.; Zhao, D.; Tenenbaum, J.; Gan, C. Prompting Decision Transformer for Few-Shot Policy Generalization. In *Proceedings of the 39th International Conference on Machine Learning (ICML), Baltimore, MD, USA, 17–23 July 2022*; PMLR: Cambridge, MA, USA, 2022; pp. 24631–24645.
22. Xie, Z.; Zhang, Y.; Zhuang, C.; Shi, Q.; Liu, Z.; Gu, J.; Zhang, G. MoDE: A Mixture-of-Experts Model with Mutual Distillation among the Experts. *Proc. AAAI Conf. Artif. Intell.* **2024**, *38*, 16067–16075.
23. Schneider, S.B.; Karl, H.; Khalili, R.; Hecker, A. Multi-Agent Deep Reinforcement Learning for Coordinated Multipoint in Mobile Networks. *IEEE Trans. Netw. Serv. Manag.* **2021**, *21*, 908–924. [CrossRef]

24. Merluzzi, M.; Filippou, M.C.; Gomes Baltar, L.; Mueck, M.D.; Calvanese Strinati, E. 6G Goal-Oriented Communications: How to Coexist with Legacy Systems? *Telecom* **2024**, *5*, 65–97. [CrossRef]
25. Sivamayil, K.; Rajasekar, E.; Aljafari, B.; Nikolovski, S.; Vairavasundaram, S.; Vairavasundaram, I. A Systematic Study on Reinforcement Learning Based Applications. *Energies* **2023**, *16*, 1512. [CrossRef]
26. Jiang, S.; Zheng, J.; Yan, F.; Zhao, S. Reinforcement-Learning-Based Network Slicing and Resource Allocation for Multi-Access Edge Computing Networks. *IEEE Trans. Cognit. Commun. Netw.* **2023**, *10*, 1132–1145. [CrossRef]