IEEE INFOCOM 2026 - IEEE Conference on Computer Communications | 979-8-3315-4961-9/26/$31.00 ©2026 IEEE | DOI: 10.1109/INFOCOM59046.2026.11571667

Faster, Smaller, and Smarter: Task-Aware Expert Merging for Online MoE Inference

Ziyi Han*, Xutong Liu†, Ruiting Zhou‡, Xiangxiang Dai*, John C.S. Lui*,
*The Chinese University of Hong Kong, †University of Washington, ‡Southeast University
Email: {zyhan24, xxdai23, cslui}@cse.cuhk.edu.hk, xutongl@uw.edu, ruitingzhou@seu.edu.cn.

***Abstract*—Sparse Mixture of Experts (SMoE) has become a preferred architecture for scaling Transformer capacity without increasing computational cost, as it activates only a small subset of experts for each input. However, deploying such an approach for *online inference* remains challenging due to the large size of a full SMoE model and the complexity of expert routing, especially in resource-constrained edge networks. Moreover, during the online inference, task information is often unavailable, making the task-level routing error-prone. In this work, we propose a novel tree-structured adaptive neural bandit router, `Tanbr`, to enable efficient and reliable online MoE inference. Instead of relying on explicit task tags, `Tanbr` estimates the task distribution over time from historical data and uses it to guide task-aware expert merging within a given pre-trained MoE. To handle the large continuous space of merging weights, `Tanbr` employs a binary tree to progressively partition the space and generate finer candidate weights. It then applies a neural bandit to learn the non-linear mapping from merging weights to model performance and decides optimal expert merging. We prove that `Tanbr` achieves a sublinear regret bound of $\mathcal{O}(\sqrt{T}\log(T))$ over T rounds, despite operating over a continuous decision space, matching regret bounds compared to existing methods. Extensive experiments show that `Tanbr` reduces inference latency by at least 45% and memory usage by up to 25%, while maintaining a high accuracy compared to many state-of-the-art methods.**

I. Introduction

Modern deep learning models, particularly large language models (LLMs), have shown remarkable performance in a variety of tasks [1]–[3]. However, this advancement has come at the cost of increasing computational demands as model capacity has grown exponentially by approximately $10\times$ each year [4]. To address these scalability challenges, Sparse Mixture of Experts (SMoE) has emerged as a promising architecture. By routing inputs to a small subset of experts, SMoE models efficiently reduce training costs while improving accuracy for tasks such as language modeling [5], machine translation [6], and image classification [7]. Beyond training efficiency, the inference efficiency of MoE has received increasing attention due to the widespread deployment of large pre-trained models [8], [9]. To this end, task-level routing has emerged as a promising strategy. It assigns entire tasks to specific experts based on task tags, improving both training and inference efficiency [10]. Compared to token-level routing, this approach reduces the frequency of router activations and expert switching, thereby lowering inference latency. As shown in Tab. I (line 1 *vs.* 2 and 3 *vs.* 4), it achieves an efficiency gain of about 35%.

TABLE I: Comparison of various routing methods with T5-based MoE. All settings are the same as described in Sec V.

Methods	**Model Size** full / loaded	**Inference Latency (s)** active 1/2/3 experts
1. Token-SMoE [5]	1.0B / 1.0B	1.81 / 2.42 / 3.05
2. Task-SMoE [10]	1.0B / 1.0B	1.16 / 1.78 / 2.41
3. Token-Merging [11]	1.0B / 1.0B	193.88
4. Task-Merging [12]	1.0B / 1.0B	3.01
5. `Tanbr` (Ours)	1.0B / 220M	0.61

However, task-routing MoE under the *online inference* setting faces several challenges that limit its adoption. *First*, varying input task distributions may lead to workload imbalance, where certain experts are activated more often than others. Replicating popular experts can alleviate this issue, but it introduces additional memory and synchronization overhead. *Second*, while selecting multiple experts can improve performance [12], it also significantly increases computational cost, as reflected in the increased inference latency in Tab. I (lines 1-2). *Third*, to avoid the overhead of loading and switching experts, the full MoE model is typically preloaded into memory. This leads to high memory demand and poses challenges for deployment in resource-limited environments, such as edge networks. *Finally*, task-routing MoE relies on the availability of task information (*e.g.*, tags) to identify the task type and route inputs to suitable experts. However, such information is often unavailable during online inference, making task-based routing difficult to apply.

To address the first two challenges, a promising approach is to merge experts by computing a weighted average of their parameters, so that the system runs a single merged expert instead of activating a subset of experts [11], [13]. This strategy incurs only the cost of running a single merged expert while leveraging the knowledge of multiple experts and prevents workload imbalance. However, existing expert merging methods either operate at the token level [11], incurring significant merging overhead (as shown in line 3 of Tab. I), or rely on task tags [12] or labeled datasets [14], limiting their applicability in the online setting. Additionally, some methods bypass explicit task tags by training classifiers to infer task types from input features [15]. However, training and deploying such classifiers

Corresponding author: Xutong Liu and Ruiting Zhou.

The work of Ruiting Zhou was supported in part by National Key Research and Development Program of China under Grant No. 2024YFB2907100, Shenzhen Science and Technology Program under Grant No. KJZD20240903100814018, National Natural Science Foundation of China under Grant No. 62232004. The work of Xiangxiang Dai was supported by the National Natural Science Foundation of China (625B2163). The work of John C.S. Lui was supported in part by the RGC GRF-14202923.

may incur substantial computational cost, limiting efficiency gains in resource-constrained online settings. These challenges lead us to ask the following research question:

How to design a router that can effectively guide task-level expert merging within a given pre-trained MoE for online inference, even when explicit task tags are unavailable?

While explicit task tags are often unavailable during inference, it is still possible to learn merging strategies by observing system feedback over time. This motivates the adoption of *an online learning approach*. In this paper, we present `Tanbr`, a novel neural bandit framework with continuous decision spaces of weights to guide expert merging for efficient and reliable online MoE inference. To the best of our knowledge, this is the *first work* to provide interoperable online MoE inference with formal theoretical guarantees. Rather than relying on specific task tags, `Tanbr` captures task characteristics by using the task distribution within each time slot, which can be easily obtained in online environments using lightweight tools such as Count-Min Sketch [16]. Using the task distribution, `Tanbr` performs expert merging once per time slot, enabling a single merged model to serve all tasks within that slot. This reduces the overhead of frequent merging and lowers memory usage. We adopt the Multi-Armed Bandit (MAB) theory to design the router for online learning. Unlike standard MAB settings, the router must handle a *continuous and multi-dimensional* decision space of merging weights and a *non-linear* mapping from merging weights to model performance. To overcome this, we propose a tree-structured partitioning method that incrementally refines the continuous decision space while controlling search complexity. We further develop a neural bandit that models the multi-dimensional reward and applies the Upper Confidence Bound (UCB) strategy to decide the optimal merging weight. Our contributions are:

- **Novel Model.** We formulate router design as a contextual bandit problem over a continuous space of merging weights. The context is the task distribution at each time slot, and the reward function can capture both linear and non-linear effects of the merging weights on model performance.
- **Tree-structured Adaptive Neural Bandit Router**. To efficiently explore the continuous decision space, `Tanbr` uses a hierarchical partition tree to generate candidate merging weights. When a node is explored enough times, it expands to provide finer candidate merging weights. `Tanbr` then utilizes a neural bandit that maintains a neural network to estimate the rewards of these candidate merging weights and selects the optimal one. Through rigorous theoretical analysis, we formally prove that `Tanbr` achieves a sublinear regret bound of $\mathcal{O}(\tilde{d}\sqrt{T}\log(T))$, where $\tilde{d}$ denotes the effective dimension of the neural tangent kernel (NTK) matrix for the neural bandit network and T is the total number of system time slots. Unlike previous neural bandit methods limited to discrete decisions, `Tanbr` extends to a large continuous decision space and still achieves comparable regret bounds.
- **Evaluation Performance**. Extensive experiments show that: i) `Tanbr` exhibits robust offline/online learning abilities, converging rapidly and performing well on metrics like loss and accuracy; ii) `Tanbr` reduces memory usage by up to 25% and improves inference efficiency by at least 45% while maintaining the same or even better accuracy compared to seven state-of-the-art baselines.

II. Related Work

Router Design in MoE. The router is a key component in all MoE architectures, controlling both the selection of expert computations and the combination of their outputs. To reduce computational and memory costs, two main types of router designs have been explored: SMoE and soft expert merging. SMoE was first introduced by Shazeer *et al.* [17]. It improves model capacity and computational efficiency by activating only a small subset of experts for each input. Since then, many works have aimed to enhance the routing function to improve scalability, stability, and performance [5], [18]–[20]. For fine-tuning and inference efficiency, Task-MoE [10] reduces inference costs by assigning entire tasks to a fixed subset of experts. Liu *et al.* [8] propose a Bayesian optimization framework to efficiently learn expert selection strategies for distributed MoE inference in serverless environments. To address gradient computation challenges during training, Muqeeth *et al.* [11] propose soft expert merging methods, where the router outputs weights to merge expert parameters. The merged parameters are then used for forward computation and backpropagation to update the expert models. Extending this idea, Lory [13] incorporates input similarity at the sequence level to guide expert merging. To improve inference efficiency, He *et al.* [12] merge experts into a unified model based on task-specific tags. Li *et al.* [14] design expert grouping and merging strategies that require substantial training data to assess expert similarity and utilization. In this paper, we focus on online inference efficiency, *i.e.*, latency and memory usage, by introducing a task-aware expert merging framework. Unlike previous approaches that rely on fixed routing or need task tags and labeled datasets, our framework uses an online approach to deal with changing task distribution without requiring explicit task tags.

Model Merging in Language Models. Recent progress in model merging for language models shows that merging separately trained models can produce a single model with strong multi-task performance, without the need to access the original data or large computational resources [21]. Works [22], [23] enable model merging by using task vectors that represent the parameter differences between pre-trained and fine-tuned models. RegMean [24] shows that for model merging, linear layers admit closed-form solutions based on training data statistics. Works [25], [26] leverage the Fisher information matrix to estimate parameter importance during merging. Instead of relying on fixed datasets or static merging strategies, we design a task-aware expert merging framework that can adaptively adjust merging decisions based on observed task distribution for online inference.

Neural Bandit. Neural contextual bandits use neural networks to model complex reward functions, allowing for both linear and non-linear relationships [27], [28]. Authors in [29],

[30] provide regret guarantees for UCB- and Thompson Sampling-based algorithms using fully connected networks. EE-Net [31] introduces an additional network to enhance exploration in neural bandits. Works [32] extend neural bandits to combinatorial actions, enabling the selection of multiple items. In this paper, we frame the router design problem as optimizing a complex reward over a *continuous decision space*. Unlike most prior neural bandit methods that focus on discrete actions, our framework handles a continuous, multi-dimensional decision space, allowing for flexible and adaptive expert merging during online inference.

III. System Model

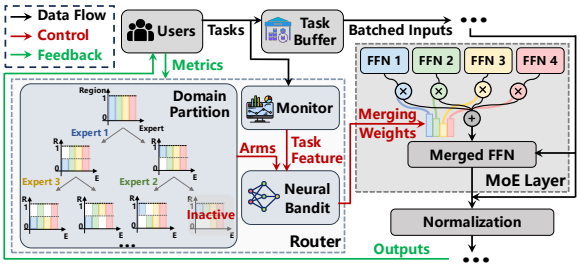


Fig. 1: Overview of `Tanbr` architecture.

A. Preliminaries

Sparse Mixture of Experts. SMoE architectures enhance the computational efficiency and scalability of neural models, such as Transformers, by replacing feed-forward network (FFN) layers with MoE layers [33]. Each MoE layer is composed of K experts ($K \geq 2$), with a router responsible for selecting the subset of experts for activation. It is assumed that all experts within a same layer share an identical FFN architecture [1]. For simplicity of presentation, we focus on a single MoE layer first, where each expert is parameterized as $E(\cdot; \boldsymbol{W}_k), \forall k \in \mathcal{K}$, with $E : \mathbb{R}^d \to \mathbb{R}^d$ representing a single expert module, and $\mathcal{K}$ denoting the set of K experts. For each input u, the router weights, denoted as $\boldsymbol{x} = [x_k]_{k\in\mathcal{K}}^\top$ are used to linearly combine the outputs of the selected experts (with $x_k = 0$, meaning expert k is not activated). Specifically, the output is computed as: $\boldsymbol{y} = \sum_{k\in\mathcal{K}} x_k E(u; \boldsymbol{W}_k)$.

Note that SMoE can operate at multiple levels, such as token [5], sequence [13], and task [10]. Specifically, expert selection can be based on a token c ($\boldsymbol{x} = \text{Router}(c)$), or a sequence of tokens $[c_i]_{i\in\mathcal{I}}^\top$ ($\boldsymbol{x} = \text{Router}(\frac{1}{I}\sum_{i\in\mathcal{I}} c_i)$), or a task tag ($\boldsymbol{x} = \text{Router}(\text{task\_tag})$), where $\text{Router} : \mathbb{R}^V \to \mathbb{R}^K$ denotes the router network/function. To improve training and inference efficiency, many studies use task-level routing strategies to train SMoE models [10], [12] or use task-level routing SMoE by merging independently trained LLMs [34]. However, SMoEs often suffer from load imbalance, and their computational cost increases significantly with the number of activated experts, thereby limiting scalability in real-world deployment.

Expert Merging. To overcome the difficulty that discrete routing decisions hinder gradient backpropagation during training, expert merging methods are proposed, which compute a weighted combination of all FFNs [11]. Specifically, given an input u, the router outputs merging weights $\boldsymbol{x}$, which are used to merge expert parameters rather than expert selection. The output is: $\boldsymbol{y} = E(u; \boldsymbol{W}), \boldsymbol{W} = \sum_{k\in\mathcal{K}} x_k \boldsymbol{W}_k$. Applying the merged expert for inference enables the system to leverage the "combined" knowledge of multiple experts while keeping the computational cost of a single expert. Moreover, this method avoids uneven workload distribution across experts.

B. Task-Aware Online Inference with MoE

Online System Model. As illustrated in Fig. 1, we assume that the service provider is equipped with a pre-trained MoE model and aims to support V different types of tasks, where $V \in \mathbb{Z}^+$ [2]. We focus on how to perform task-level routing during online inference because it can improve efficiency by reducing router activations and expert switching [10]. When deployed for inference, the system operates in an *online environment*, where requests arrive randomly. Although each task is associated with a type, this information is typically unavailable to the router during online inference due to the absence of explicit task tags or labels. Consequently, traditional task-level routing methods [10], [12] that rely on this information are not effective. To address this, we introduce a *monitor* module (depicted in Fig. 1), which estimates the current task distribution based on historical data and lightweight prediction tools [35], [36], such as exponential moving average, Count-Min Sketch [16]. Specifically, at each time slot t (e.g., every 30 minutes or longer), the monitor generates a task feature vector $\boldsymbol{\psi}_t = [\psi_{t,v}]_{v\in\mathcal{V}}^\top$, where $\psi_{t,v}$ denotes the estimated proportion of task type v during that slot.

Core Issue. To further reduce computational and memory overhead during inference, we also adopt expert merging strategy to serve inference tasks. The main challenge is to design and train a new router that can effectively leverage task features at each time slot to guide expert merging within the pre-trained MoE, without requiring explicit task tags. Note that our router must dynamically adapt to evolving task distributions to ensure efficient and reliable online inference.

Router Learning Objective. There are two situations: **i) Fine-tune before online inference**: Although expert merging is resource-efficient, directly applying merged models for online inference can result in performance drops, especially for post-layer normalization architectures (e.g., T5-based MoEs [2]). They are sensitive to parameter shifts and therefore require fine-tuning before online adaptation. This fine-tuning phase is performed with the labeled training data, on which the system can calculate task-specific losses (e.g., Mean Squared Error or Cross-Entropy) to fine-tune the model and train the router. For each time slot t, the loss vector is denoted as $\boldsymbol{l}_t = [l_{t,v}]_{v\in\mathcal{V}}^\top$, where $l_{t,v}$ represents the average loss of task type v during that time slot. During the fine-tuning phase, the goal is to minimize the overall loss. Therefore, we define the reward of the router

[1] In cases of structural discrepancies, weight permutation alignment techniques can be employed to mitigate mismatches among neurons [14].

[2] In this work, we assume that task-level routing was used during training, as it offers better compatibility. However, our method is still applicable even if token-level or sequence-level routing was used during training.

as the negative value of the loss, *i.e.*, $\boldsymbol{r}_t = -\boldsymbol{l}_t$. **ii) Online learning during inference**: Architectures that use pre-layer normalization (such as BERT-based MoEs [1]) show stronger robustness when merging experts, allowing the merged model to be used directly for inference. In this setting, the reward can be directly defined using performance metrics (*e.g.*, accuracy) observed during inference: $\boldsymbol{r}_t = \boldsymbol{a}_t, \boldsymbol{a}_t = [a_{t,v}]_{v\in\mathcal{V}}^\top$, where $a_{t,v}$ denotes the average reward of task type v during time slot t. This removes the need for labeled data and allows the router to be trained fully in an online manner. *More generally*, the reward can be abstracted as a function of the merging weights $\boldsymbol{x}_t$ at each time slot t, *i.e.*, $\boldsymbol{r}_t = f^*(\boldsymbol{x}_t)$, where $f^* : \mathbb{R}^K \to \mathbb{R}^V$ denotes the non-linear mapping from the merging weight to the reward, omitting intermediate steps such as expert merging, model execution, and loss/accuracy calculation. The total reward is computed as the weighted sum $\boldsymbol{\psi}_t^\top \boldsymbol{r}_t$, which is linear in $\boldsymbol{r}_t$.

C. Problem Formulation

Mathematical Formulation. We formulate the problem as an online constrained learning optimization problem, aiming to maximize the cumulative reward over a finite time horizon $\mathcal{T}$, as follows,

$$\max \textstyle\sum_{t\in\mathcal{T}} \boldsymbol{\psi}_t^\top \boldsymbol{r}_t, \tag{1}$$

$$\text{s.t.} \quad \textstyle\sum_{k\in\mathcal{K}} x_{t,k} = 1, \forall t \in \mathcal{T}. \tag{1a}$$

$$\textstyle\sum_{k\in\mathcal{K}} \Pi(x_{t,k} > 0) \le B, \forall t \in \mathcal{T}, \tag{1b}$$

$$\boldsymbol{x}_t \in \mathcal{X}, \forall t \in \mathcal{T}, \tag{1c}$$

where $\mathcal{X}$ is a measurable space of merging weights, *i.e.*, $\mathcal{X} = [0,1]^K$; $\Pi(\cdot)$ is an indicator function, which returns 1 if the input condition is satisfied and 0 otherwise. Constraint (1a) ensures that the sum of merging weights produced by the router is equal to 1. Constraint (1b) limits the number of selected experts to at most B, a predefined constant that reflects available computational and memory resources.

Challenges. i) Problem (1) can be reduced to the well-known Knapsack Problem [37], [38], classifying it as an NP-hard problem. ii) The decision variables are both multi-dimensional and continuous, leading to high computational costs when searching for an optimal solution. iii) The reward acquisition is context-dependent and follows a non-linear mapping, which introduces additional complexities. iv) To preserve valuable information and utilize the problem's linear components, the reward is multi-dimensional, which increases learning complexity and requires balancing multiple objectives.

IV. Algorithm Design

A. Main Idea

Given the challenges of Problem (1), reinforcement learning (RL) is less suitable, due to the absence of long-horizon state transitions [39]. We therefore adopt an MAB framework, which offers theoretical performance guarantees and greater interpretability compared to end-to-end learning methods [40], [41]. However, existing MAB methods are inadequate due to the presence of both non-linear parameters and a continuous decision space. Here, we introduce a novel framework, tree-structured adaptive neural bandit router, `Tanbr`, for fine-tuning and online inference within a pre-trained MoE. `Tanbr` includes the following steps:

i) **Tree Partitioning.** An infinite binary cover tree is used to discretize the *continuous decision space*, where each decision corresponds to a feasible merging weight. The tree structure allows for the systematic generation of candidate merging weights that satisfy the given constraints.

ii) **Neural Bandit Framework.** To handle the *non-linear dependencies* of reward, a neural bandit approach is employed. It utilizes the power of deep neural networks to learn the mapping from merging weights to rewards, while the UCB method [29] guides the selection of merging weights by balancing exploration and exploitation. The network's output layer generates *multi-dimensional reward* estimates, allowing the merged expert to effectively support multiple tasks.

iii) **Incremental Learning and Exploration.** The selected decision guides the merging of experts within the pre-trained MoE for fine-tuning/inference, the observed reward is then used to update the parameters of `Tanbr`. As exploration progresses, the tree is incrementally expanded to generate a finer set of candidate merging weights for improved decision-making.

B. Tree-Structured Measurable Space Partitioning

To effectively explore the continuous and multi-dimensional decision space of merging weights, we use a tree-structured partitioning method [42], which narrows promising regions while keeping the search complexity manageable. The tree structure also enables the pruning of subregions with unpromising merging weights, reducing unnecessary exploration.

Partition Tree Design. We adopt a pre-defined partition tree $\mathcal{H} := \{\mathcal{H}_{h,i}\}$ over the measurable space $\mathcal{X}$ to facilitate the optimization process. Each node $\mathcal{H}_{h,i}$ represents a subregion within the space $\mathcal{X}$ where h denotes the depth and i the index at that depth. As illustrated in the *domain partition* module in Fig. 1, the tree recursively partitions the decision space in a binary and non-overlapping manner, following the rule: $\mathcal{H}_{0,1} = \mathcal{X}$, $\mathcal{H}_{h,i} = \mathcal{H}_{h+1,2i-1} \cup \mathcal{H}_{h+1,2i}$. At the start of the algorithm, the tree $\mathcal{H}$ is initialized with only the root node $\mathcal{H}_{0,1} = \mathcal{X}$. At each time slot t, a candidate merging weight is selected from a discrete set sampled from all current *active leaf nodes*. The number of times node (h,i) has been selected up to time t is denoted as $P_{h,i}(t)$. Once this count exceeds a depth-dependent threshold $\tau_h(t)$, the corresponding node is expanded into two child nodes, with the partitioning dimension randomly selected. The threshold is defined as,

$$\tau_h(t) = \frac{C^2 \log(t/\delta)}{\nu_1^2} \rho^{-2h}, \tag{2}$$

where C is an fixed constant, ν_1 and ρ are smoothness parameters (as specified in Assumption 1), and δ is the confidence parameter. Higher thresholds encourage more exploration before partitioning, while lower thresholds lead to earlier node expansion but may result in insufficient sampling. Adjusting the threshold parameter helps balance fine-grained partitioning with adequate exploration at each node.

Constraint Satisfaction. Given the set of active leaf nodes, candidate merging weights are generated by identifying points

within their respective cover region that satisfy the *constraint (1a)*. A leaf node is considered *active* if its cover region contains at least one feasible merging weight; otherwise, it is marked *inactive* and excluded from future selection and expansion. For example, if a leaf node has a cover region $[[0.5, 1], [0.75, 1]]$, and no point within this region satisfies the constraint (1a), the node is marked as inactive. The search for feasible merging weights within a region can be formulated as a linear programming (LP) problem. This problem can be efficiently solved using standard optimization methods, such as the simplex method or interior-point algorithms [43]. To satisfy the constraint (1b), the merging weights are adjusted by retaining the B largest values, normalizing them, and setting the rest to zero.

C. Design of `Tanbr`

Neural Network for Reward Prediction. To capture the non-linear uncertainty in the reward function, we employ a fully connected neural network to learn and predict the reward function $f^*(\boldsymbol{x})$, and reformulate it as,

$$f(\boldsymbol{x}, \boldsymbol{\theta}) = \sqrt{w}\boldsymbol{\theta}_L \sigma(\boldsymbol{\theta}_{L-1}\sigma(\dots\sigma(\boldsymbol{\theta}_1\boldsymbol{x}))), \tag{3}$$

where w is the width of each hidden layer, L denotes the depth of the network, and $\sigma(\cdot)$ is the activation function. The output layer has a dimension of V to predict multi-dimensional rewards for the V different task types. The parameter vector of the network is defined as $\boldsymbol{\theta} = [\boldsymbol{\theta}_1^\top, \dots, \boldsymbol{\theta}_L^\top]^\top \in \mathbb{R}^p$, where $p = wK + wV + w^2(L-1)$. Denote the gradient of $f(\boldsymbol{x}, \boldsymbol{\theta})$ as $\boldsymbol{g}(\boldsymbol{x}, \boldsymbol{\theta}) = \nabla_{\boldsymbol{\theta}} f(\boldsymbol{x}, \boldsymbol{\theta}) \in \mathbb{R}^p$.

Our proposed tree-structured adaptive neural bandit router, `Tanbr` is presented in Alg. 1, as follows:

Initialization. `Tanbr` begins by initializing the neural network, where each element of the parameter vector $\boldsymbol{\theta}$ is randomly generated from an appropriate Gaussian distribution [29]. Furthermore, the partition tree is initialized with a root node that spans the entire decision space, *i.e.*, $\mathcal{X} = [0, 1]^K$.

Decision Making. At each time slot t, `Tanbr` gets the task feature vector $\boldsymbol{\psi}_t$ and extracts the set of active leaf nodes $\mathcal{N}_t$ from the partition tree $\mathcal{H}$ (lines 3-4). For each active node, a feasible merging weight is generated that satisfies both its cover region and constraint (1a), forming the candidate set (line 5). Then, for each merging weight $\boldsymbol{x}_{h,i}$, line 6 calculates its UCB $U_{h,i}$ by,

$$\begin{aligned} U_{h,i} =& \boldsymbol{\psi}_t^\top f(\boldsymbol{x}_{h,i}, \boldsymbol{\theta}_{t-1}) + \nu_1 \rho^h + \\ & \gamma_{t-1}\sqrt{\boldsymbol{g}(\boldsymbol{x}_{h,i}, \boldsymbol{\theta}_{t-1})^\top \boldsymbol{Z}_{t-1}^{-1} \boldsymbol{g}(\boldsymbol{x}_{h,i}, \boldsymbol{\theta}_{t-1})/w}, \end{aligned} \tag{4}$$

where $\boldsymbol{Z}$ is an auxiliary variable; γ_t is a positive scaling factor and is defined as $\gamma_t = \sqrt{1 + C_1 w^{-1/6}\sqrt{\log w} L^4 t^{7/6}\lambda^{-7/6}} \cdot (\upsilon\sqrt{\log(\det \boldsymbol{Z}_t / \det \lambda \boldsymbol{I}) + C_2 w^{-1/6}\sqrt{\log w} L^4 t^{5/3}\lambda^{-1/6} - 2\log\delta} + \sqrt{\lambda} S) + (\lambda + C_3 tL)[(1 - \eta w\lambda)^{J/2}\sqrt{t/\lambda} + w^{-1/6}\sqrt{\log w} \cdot L^{7/2} t^{5/3}\lambda^{-5/3}(1 + \sqrt{t/\lambda})]$, where λ denotes the regularization parameter, S is the norm parameter, and η represents the step size for updating the parameters of the neural network. The first term estimates the expected reward, the second reflects the granularity of the node's region in the tree, and the third term quantifies the uncertainty in the reward prediction. Line 7 selects merging weight $\boldsymbol{x}_{h_t,i_t}$ with the highest upper bound.

Parameter Updating. The selected merging weight $\boldsymbol{x}_{h_t,i_t}$ is used to merge experts, and the merged model is deployed for fine-tuning/inference. Afterward, the corresponding reward $f^*(\boldsymbol{x}_{h_t,i_t})$ is observed (line 8). `Tanbr` then updates the selection count $P_{h,i}$ for all active leaf nodes in $\mathcal{N}_t$ (line 9). If the selected node has been explored sufficiently, *i.e.*, the number of times the node has been selected $P_{h_t,i_t}(t)$ exceeds the threshold $\tau_{h_t}(t)$, it is expanded (lines 10-12). As described in Sec. IV-B, the expansion is performed by randomly selecting a dimension of the region for binary partitioning, which results in two new leaf nodes being added to $\mathcal{H}$. At the end of t, line 13 updates the auxiliary variable $\boldsymbol{Z}_t$ and line 14 updates the neural network parameters $\boldsymbol{\theta}_t$ by minimizing the loss function $L(\boldsymbol{\theta})$ using stochastic gradient descent (SGD) with a step size η. The loss function we used is: $L(\boldsymbol{\theta}) = \sum_{s=1}^{t} \frac{1}{2}\left(f(\boldsymbol{x}_{h_s,i_s}, \boldsymbol{\theta}) - f^*(\boldsymbol{x}_{h_s,i_s})\right)^2 + \frac{\lambda}{2}\|\boldsymbol{\theta} - \boldsymbol{\theta}_0\|_2^2$, where λ denotes the regularization parameter.

Algorithm 1 Tree-structured adaptive neural bandit router

1: **Initialization:** $\boldsymbol{\theta}_0$, $\boldsymbol{Z}_0 = \lambda \boldsymbol{I}$, $\mathcal{H} = \{[0, 1]^K\}$.
2: **for** $t \in \mathcal{T}$ **do**
3: Get the task feature vector $\boldsymbol{\psi}_t$ from monitor;
4: Get the set of active leaf nodes $\mathcal{N}_t$ from the tree $\mathcal{H}$;
5: Generate the candidate set $\{\boldsymbol{x}_{h,i}\}_{(h,i)\in\mathcal{N}_t}$ using optimization methods;
6: Compute $U_{h,i}$ for all $\{\boldsymbol{x}_{h,i}\}_{(h,i)\in\mathcal{N}_t}$ using Eq. (4);
7: Let $(h_t, i_t) = \arg\max_{(h,i)\in\mathcal{N}_t} U_{h,i}$;
8: Apply $\boldsymbol{x}_{h_t,i_t}$ to merge experts for fine-tuning/ inference and then observe reward $f^*(\boldsymbol{x}_{h_t,i_t})$;
9: Update selection count: $P_{h_t,i_t}(t) = P_{h_t,i_t}(t-1) + 1$; $P_{h,i}(t) = P_{h,i}(t-1), \quad \forall (h,i) \in \mathcal{N}_t / (h_t, i_t)$;
10: **if** $P_{h_t,i_t}(t) \geq \tau_{h_t}(t)$ **then**
11: $\mathcal{H} = \mathcal{H} \cup expand(h_t, i_t)$;
12: **end if**
13: Update auxiliary variable with gradient: $\boldsymbol{Z}_t = \boldsymbol{Z}_{t-1} + \boldsymbol{g}(\boldsymbol{x}_{h_t,i_t}, \boldsymbol{\theta}_{t-1}) \cdot \boldsymbol{g}(\boldsymbol{x}_{h_t,i_t}, \boldsymbol{\theta}_{t-1})^\top / w$;
14: Update: $\boldsymbol{\theta}_t = \boldsymbol{\theta}_{t-1} - \eta \nabla L(\boldsymbol{\theta}_{t-1})$ with J steps;
15: **end for**

D. Theoretical Analysis

To facilitate a more concise and structured theoretical analysis, we augment the neural network with an additional linear output layer parameterized by $\boldsymbol{\psi}_t$. Let the complete set of parameters be denoted by $\boldsymbol{\xi} = [\boldsymbol{\theta}^\top, \boldsymbol{\psi}_t^\top]^\top$. Accordingly, the estimated and real total reward is expressed as $f(\boldsymbol{x}, \boldsymbol{\xi})$ and $f_1^*(\boldsymbol{x})$. All detailed proofs are provided in the Appendix.

Candidate Arms Upper Bound. We first write some assumptions [42].

Assumption 1. *i)* **(Dissimilarity)** *The space $\mathcal{X}$ is equipped with a dissimilarity function $\ell : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ such that $\ell(\boldsymbol{x}, \boldsymbol{x}') \geq 0$ for all $\boldsymbol{x}, \boldsymbol{x}' \in \mathcal{X}$, and $\ell(\boldsymbol{x}, \boldsymbol{x}) = 0$. ii) Define the diameter of a subset $\mathcal{A} \subseteq \mathcal{X}$ as $diam(\mathcal{A}) = \sup_{\boldsymbol{x},\boldsymbol{y}\in\mathcal{A}} \ell(\boldsymbol{x}, \boldsymbol{y})$, and the ℓ-open ball of radius $\epsilon > 0$ and center $\boldsymbol{x} \in \mathcal{X}$ as $\mathcal{B}(\boldsymbol{x}, \epsilon) = \{\boldsymbol{x}' \in \mathcal{X} : \ell(\boldsymbol{x}, \boldsymbol{x}') \leq \epsilon\}$. iii)* **(Local Smoothness)** *We assume that there exist constants $\nu_1, \nu_2 > 0$ and $0 < \rho < 1$ such that for all nodes*

(h,i): *a)* $diam(\mathcal{H}_{h,i}) \leq \nu_1\rho^h$*; b) There exists* $\boldsymbol{x}^o_{h,i} \in \mathcal{H}_{h,i}$ *such that:* $\mathcal{B}_{h,i} = \mathcal{B}(\boldsymbol{x}^o_{h,i}, \nu_2\rho^h) \subseteq \mathcal{H}_{h,i}$*; iii)* $\mathcal{B}_{h,i} \cap \mathcal{B}_{h,j} = \emptyset$ *for all* $i \neq j$*, iv) For all* $\boldsymbol{x} \in \mathcal{X}$*,* $f^* - f(x) \leq \ell(\boldsymbol{x}^*, \boldsymbol{x})$.

The dissimilarity assumption measures the difference between any two points in space. The local smoothness assumption ensures that the space is smoothly partitioned: regions shrink as depth increases, are non-overlapping, and the function behaves smoothly within each region.

Lemma 1. *Given the threshold* $\tau_h(t)$ *defined in Eq. (2), the number of candidate merging weights at* $\forall t$ *is bounded by:*

$$N_t \leq N = 2^{1/(1-\rho)}T\nu_1^2 / \left(C^2 \log(T/\delta)\right) = \mathcal{O}(T). \tag{5}$$

Regret Upper Bound. The regret of an algorithm is defined as the difference between the total expected reward achieved by the optimal offline solution and that obtained by the algorithm. Let x_t^* be the merging weight with the highest reward, *e.g.*, $x_t^* = \boldsymbol{\psi}_t \arg\max_{\boldsymbol{x}\in\mathcal{X}} f^*(\boldsymbol{x})$. The regret of `Tanbr` is defined as follows: $R = \sum_{t\in\mathcal{T}} R_t, \quad R_t = \mathbb{E}\left[f_1^*(\boldsymbol{x}_t^*) - f_1^*(\boldsymbol{x}_{h_t,i_t})\right]$. Let $\mathcal{X}_T = [\boldsymbol{x}_{h,i}]^\top_{(h,i)\in\mathcal{N}_t, t\in\mathcal{T}}$ be the complete set of candidate merging weights selected up to time T. By Lemma 1, we have $|\mathcal{X}_T| \leq TN$. We first introduce the following assumption.

Assumption 2. *For any* $\boldsymbol{x} \in \mathcal{X}_T$*, it satisfies* $\|\boldsymbol{x}\|_2 = 1$ *and* $[\boldsymbol{x}]_j = [\boldsymbol{x}]_{j+V/2}$*, for* $1 \leq j \leq V/2$*. Furthermore, for some* $\lambda_0 > 0$*, the NTK matrix* $\boldsymbol{M}$ *satisfies* $\boldsymbol{M} \succeq \lambda_0 \boldsymbol{I}$.

This assumption is commonly used in neural bandit research [29]. The constraint $\|\boldsymbol{x}\|_2 = 1$ is introduced to normalize the context vector. The symmetry condition enables the construction of an equivalent context vector in the form $\boldsymbol{x}' = [\boldsymbol{x}^\top, \boldsymbol{x}^\top]^\top/\sqrt{2}$.

Theorem 1. *Let* $\tilde{d}$ *be the effective dimension of the NTK matrix* $\boldsymbol{M}$ *and* $\boldsymbol{m} = [f_1^*(\boldsymbol{x})]^\top_{\boldsymbol{x}\in\mathcal{X}_T}$ *as the true reward vector of the candidate merging weights. There exist constants* $C_1, C_2 > 0$ *such that for any* $\delta \in (0,1)$*, if* $w \geq \text{poly}(T, L, K, \lambda^{-1}, \lambda_0^{-1}, S^{-1}, \log(1/\delta))$*,* $\eta \leq C_1(TdL + d\lambda)^{-1}$*,* $\lambda \geq \max\{1, S^{-2}\}$*, and* $S \geq \sqrt{2\boldsymbol{m}^\top \boldsymbol{M}^{-1}\boldsymbol{m}}$*, then with probability at least* $1-\delta$*, the regret of Alg. 1 satisfies,*

$$\begin{aligned} R \leq & 3\sqrt{T}\sqrt{\tilde{d}\log(1+TN/\lambda)+2}\cdot \\ & \Big[\upsilon\sqrt{\tilde{d}\log(1+TN/\lambda)+2-2\log\delta} \\ & +(\lambda+C_3TL)(1-\eta w\lambda)^{J/2}\sqrt{T/\lambda}+2\sqrt{\lambda S}\Big]+T+1. \end{aligned} \tag{6}$$

Remark. Following Remark 4.7 in [29], by setting $J = \mathcal{O}(TL/\lambda)$ we have $(\lambda + C_3TL)(1-\eta w\lambda)^{J/2}\sqrt{T/\lambda} \leq \sqrt{\lambda S}$. By treating ν, λ, and η as constants and applying Lemma 1, the regret bound in Theorem 1 simplifies to $R_T = \mathcal{O}(\tilde{d}\sqrt{T}\log(T))$, matching the regret bounds in prior neural bandit studies [27]–[30]. Compared to LinUCB [44], this bound includes an additional factor $\tilde{d}$, reflecting the complexity of neural representations. This extra cost is acceptable given the ability to capture more expressive and nonlinear context. A key contribution of `Tanbr` is maintaining these theoretical guarantees in a continuous decision space. This is achieved through smoothness assumptions and adaptive partitioning techniques that ensure efficient exploration while avoiding exhaustive search. This result formally shows the theoretical suitability of `Tanbr` for long-term deployment in online MoE systems.

V. Performance Evaluation

A. System Implementation

MoE Models and datasets. We evaluate our `Tanbr` using both i) encoder-decoder, and ii) encoder-only Transformer models to ensure a comprehensive evaluation. For both models, every FFN layer is replaced by an MoE layer containing $K = 8$ experts with the same architecture. The encoder-decoder model is based on T5 [2], which includes 12 FFN layers in both the encoder and decoder, and has about $1.0B$ parameters. The decoder-only model is based on BERT [1] and contains 12 FFN layers with approximately $660M$ parameters. Both models are pre-trained using task routing [10], [34].

Datasets. We evaluate the models on $V = 8$ common tasks from the GLUE benchmark, including natural language inference (*i.e.*, MNLI, QNLI, RTE), sentiment analysis (*i.e.*, SST-2), sentence similarity (*i.e.*, MRPC, QQP, STS-B), and linguistic acceptability (*i.e.*, CoLA) [45], [46]. Among them, CoLA is evaluated by the Matthews correlation coefficient, STS-B is evaluated by the average value of Pearson and Spearman correlation coefficients, and the remaining tasks are evaluated by accuracy. Each dataset is divided into training and validation sets in an $80/20$ split. Both the batch size and the maximum sequence length (for source and target) are set to 128.

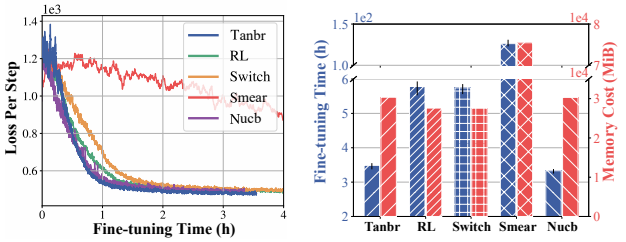
Framework Setting. In `Tanbr`, we set the smoothness parameter $\nu_1 = 1$, ρ over $\{0.3, 0.5, 0.7, 0.9\}$. The regularization parameter is $\lambda = 1$, and the exploration parameter is $\upsilon = 1$. The confidence level is set as $\delta = 1$. We use learning rate η over $\{1e-4, 1e-3, 1e-2, 1e-1\}$. The neural network has width $d = 64$ and depth $L = 2$. The maximum number of experts selected for merging is fixed to $B = 8$. All experiments are run with PyTorch on NVIDIA A100 GPUs with $40GB$ of memory.

Baselines. We compare `Tanbr` with the following methods: i) `RL` [47]: a RL approach for expert selection in sparse MoE, assigning one expert per token. ii) `Switch` [5]: a neural network-based method that learns to select one expert per token for sparse MoE routing. iii) `SMEAR` [11]: a neural network-based token-level expert merging method. iv) `NUCB` [29]: a neural bandit method that randomly samples a set of 20 feasible merging weights and selects one by neural UCB at each step.v) `Regmean` [24]: a model merging method that computes weights using a closed-form solution to the merging problem. vi) `Fisher` [26]: a model merging method that leverages Fisher information to estimate parameter importance and performs weighted merging accordingly. vii) `Average` [48]: a model merging method that performs element-wise averaging of all parameters.

B. Results with Fine-tuning and Inference

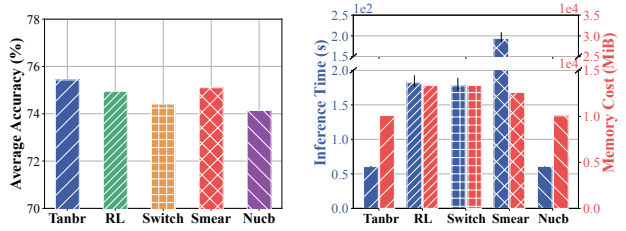
Fine-tuning Phase. As mentioned in Section III, T5-based MoE with post-layer normalization requires fine-tuning when expert merging is applied due to its sensitivity to parameter shifts. Therefore, we fine-tune its normalization layers for $10,000$ steps using the training data, while simultaneously

training a new router with different algorithms. The results are depicted in Fig. 2, showing the training loss curve, total training time, and memory usage. From the figure, we have some observations. *First*, Tanbr starts with a higher loss since it begins with fewer candidate merging weights. However, as the partition tree refines over time, Tanbr converges much faster and achieves lower final loss than all baselines. In comparison, Nucb, which selects decisions from randomly generated merging weights, exhibits more fluctuation and slower convergence. *Second*, both Tanbr and Nucb (task-level routing) achieve the *smallest training time*, achieving approximately 40% reduction compared to RL and Switch. Although Tanbr introduces a small amount of overhead due to maintaining the partition tree, it remains highly efficient. In contrast, Smear, which merges at the token level, results in a 20× increase in computational cost because it cannot use *lightweight adapters* under the given pre-trained MoE architecture, making its overhead unavoidable. *Third*, Tanbr and Nucb consume about 10% more memory than RL and Switch, as they need to maintain the loss gradients of multiple experts. Smear performs token-level merging, incurring the highest memory footprint, exceeding other methods by 179% because it stores token-wise loss gradients for all experts.



(a) Fine-tuning Loss (b) Time/Memory Cost

Fig. 2: Fine-tuning performance with T5-based MoE.

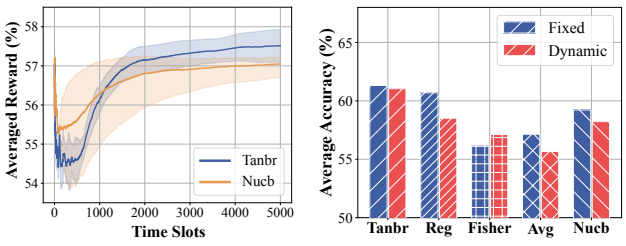


(a) Average Accuracy (b) Time/ Memory Cost

Fig. 3: Inference performance with T5-based MoE.

Online Inference. The fine-tuned models are deployed for online inference without further updates to the model parameters or router. The inference results are presented in Fig. 3, including the average performance across tasks (detailed per-task metrics are provided in Tab. II), inference time per batch, and memory usage. We highlight three main observations: *First*, Tanbr achieves the *best average performance* along with consistently better performance on individual tasks. This is because it dynamically adjusts the expert merging strategy based on online task features and fully exploits the information from multiple experts. *Second*, Tanbr demonstrates highly *efficient inference speed*, requiring only one expert merging decision and execution at the start of each time slot. This reduces inference time per batch by 65% compared to RL and Switch, nearly 99% over Smear. *Third*, Tanbr achieves the *lowest memory consumption*. Since only the merged model is loaded, it reduces memory use by 20% to 25%.

C. Results with Online Learning for Inference



(a) Online Reward (b) Average Accuracy

Fig. 4: Online Inference with BERT-based MoE.

Here, we focus on BERT-based MoE models that do not require fine-tuning and allow the router to be trained online during inference. Since loss information is unavailable during online inference, traditional router training methods cannot be applied. Thus, we compare our Tanbr with model merging methods. The online learning performance is shown in Fig. 4(a). Although Tanbr performs worse in the early stages due to the limited number of candidate merging weights, it converges faster and achieves better performance than Nucb as the partition tree becomes more refined over time. This result highlights the advantage of our tree partitioning strategy, which efficiently explores the search space during online updates. Overall, Tanbr demonstrates strong online learning capability.

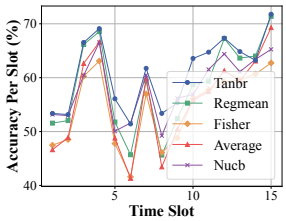


Fig. 5: Online inference per slot performance with BERT-based MoE.

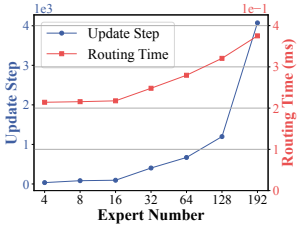


Fig. 6: Update and routing overhead of Tanbr under different system scales.

After 5000 time slots of online learning, we further evaluate the performance over additional time slots. The results under both fixed and dynamic task features are shown in Fig. 4(b) and the per-slot performance details under dynamic task features are shown in Fig. 5. We can observe that Tanbr outperforms the baselines by up to 9% under fixed task features and by $5-10\%$ under dynamic task features. The performance gain is especially notable in dynamic settings, as Tanbr adapts to evolving task distributions in real-time by merging experts based on task features. This adaptability enables it to surpass other methods that rely on static or less flexible merging strategies.

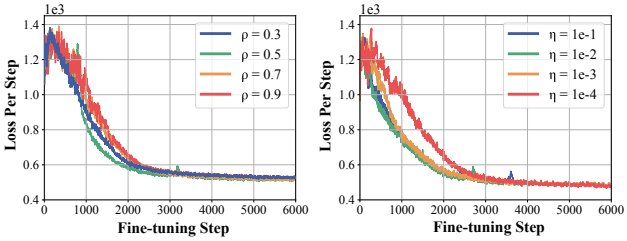
D. Scalability and Ablation study

Scalability Evaluation. To evaluate the scalability of Tanbr, we conduct experiments on MoE models with 4 to 192 experts.

TABLE II: Performance metrics for T5-based MoE with different routing methods.

Method	CoLA mat	MNLI acc	MRPC acc/f1	QNLI acc	QQP acc/f1	RTE acc	SST-2 acc	STS-B pear/spear	Avg
Tanbr	42.9	75.3	78.4/84.4	84.1	82.0/74.8	63.8	91.5	86.1/85.9	**75.43**
RL	42.1	75.7	77.0/83.3	82.6	81.6/74.2	65.3	91.2	83.6/83.8	74.95
Switch	43.2	71.5	79.3/85.6	82.8	81.6/75.2	62.3	90.4	84.3/84.1	74.41
Soft	43.8	74.0	79.9/85.9	82.9	81.5/74.6	62.8	90.6	85.8/85.7	75.10
Nucb	40.4	73.2	80.8/84.6	82.6	81.9/74.3	62.8	90.3	83.3/82.8	74.14

Fig. 6 shows the number of update steps required to identify 20 feasible leaf nodes and the time needed to compute the optimal merging weight from them. The results show that as the number of experts increases, more update steps are required due to the higher dimensionality of the decision space $\mathcal{X}$, which leads to more partitioning options. Although the routing time per decision also increases with the number of experts, it remains low (less than 0.4 ms even when $K = 192$). These results indicate that Tanbr maintains high efficiency and scalability in large-scale settings.



(a) Smoothness Parameter ρ (b) Learning Rate η

Fig. 7: Learning performance of Tanbr with T5-based MoE under different settings of parameters.

Ablation Study. We analyze the sensitivity of our Tanbr to key hyperparameters. Fig. 7 presents the learning performance of the T5-based MoE under different settings of the smoothness parameter ρ and the neural network learning rate η. A smaller ρ makes Tanbr more conservative when partitioning the cover tree, requiring more observations before a node is expanded. As seen in Fig. 7, a smaller ρ shortens the initial random phase, since the merging weights exhibit less variation and allow faster fine-tuning. However, this leads to relatively lower final performance, as the decision space is insufficiently explored. In contrast, a larger ρ allows for better exploration of the decision space but results in longer initial random times and slower fine-tuning. Therefore, choosing an appropriate value of ρ is critical for balancing exploration and exploitation. The learning rate η controls the magnitude of updates during the training of the neural network. A higher learning rate accelerates convergence but may cause instability, while a lower learning rate results in more stable training but slower progress. Tuning η is essential to achieve both stability and efficient convergence.

VI. Conclusion

In this work, we propose Tanbr, a tree-structured adaptive neural bandit router, to improve the efficiency of online inference with MoEs. By utilizing task distributions, which are easily obtained in online environments, Tanbr dynamically merges experts within a given pre-trained MoE in a task-aware manner, enabling a single merged expert to handle multiple tasks. This is achieved through a tree-structured partitioning method that efficiently explores the continuous decision space, followed by a neural bandit approach that models the non-linear mapping between merging weights and performance rewards, and selects the optimal merging weight. Through both theoretical analysis and extensive experiments, we show that Tanbr achieves sublinear regret and significantly reduces memory usage and inference latency while maintaining a high accuracy. These advantages make Tanbr particularly suitable for deploying MoE models in resource-constrained settings, such as edge networks.

To reduce interference among multiple experts [12], [14], future work could explore grouping similar experts for merging. This can be achieved by applying advanced clustering techniques [49] to dynamically form expert groups, which could improve both efficiency and accuracy across various applications.

Appendix

A. Proof of Lemma 1

Besides the assumptions in Sec. IV-D, we also have the following assumption.

Assumption 3. (Near-Optimality dimension) *Let $\epsilon = 3\nu_1\rho^h$ and $\epsilon' = \nu_2\rho^h < \epsilon$. For any subset of ϵ-optimal nodes $\mathcal{X}_\epsilon = \{\boldsymbol{x} \in \mathcal{X} : f^* - f(\boldsymbol{x}) \le \epsilon\}$, there exists a constant C such that $\mathcal{N}(\mathcal{X}_\epsilon, \ell, \epsilon') \le C(\epsilon')^d$, where d is the near-optimality dimension of f and $\mathcal{N}(\mathcal{X}_\epsilon, \ell, \epsilon')$ is the ϵ'-cover number of $\mathcal{X}_\epsilon$ with respect to the dissimilarity measure ℓ.*

Then, we characterize the complexity of the problem using the near-optimality dimension. We first establish an upper bound on the maximum depth of the partition tree in Tanbr.

Lemma 2. *Given the threshold $\tau_h(t)$ defined in Eq. (2), the depth $H(t)$ of the tree $\mathcal{H}_t$ is bounded by:*

$$H(t) \le H_{max}(T) \le \frac{1}{1-\rho} \log \frac{T\nu_1^2}{C^2 \log(T/\delta)}. \tag{7}$$

Proof. Recall that a node (h, i) is expanded only when its pull count $P_{h,i}(t)$ exceeds the threshold $\tau_h(t)$. To find the maximum possible depth $H(t)$ of any node that may be expanded up to round t, we consider the condition under which a node can be expanded: $\tau_h(t) \le t$. By substituting the expression for $\tau_h(t)$ from Eq. (2), we obtain: $\frac{C^2 \log(t/\delta)}{\nu_1^2}\rho^{-2h} \le t \Rightarrow \rho^{-2h} \le \frac{t\nu_1^2}{C^2 \log(t/\delta)}$.

Taking the logarithm on both sides, we have the depth $H(t)$ bounded by: $H(t) \leq \frac{1}{2\log(1/\rho)} \log\left(\frac{t\nu_1^2}{C^2\log(t/\delta)}\right)$. To simplify the upper bound to depend solely on T, we use the inequalities $t \leq T$ and $\log(t/\delta) \geq \log(T/\delta)$. This gives:

$$H(T) \leq \frac{1}{2\log(1/\rho)} \log(\frac{T\nu_1^2}{C^2\log(T/\delta)}) \leq \frac{1}{1-\rho} \log(\frac{T\nu_1^2}{C^2\log(T/\delta)}),$$

where the second inequality follows $2\log(1/\rho) \geq 1-\rho, \forall \rho \in (0,1)$. This completes the proof of Lemma 2.

Lemma 2 ensures that the maximum depth of the tree does not exceed $O(\log T)$. Furthermore, we derive an upper bound on the number of arms at any time slot, denoted by N as:

$$N_t \leq \frac{2^{1/(1-\rho)} t\nu_1^2}{C^2\log(T/\delta)} \leq \frac{2^{1/(1-\rho)} T\nu_1^2}{C^2\log(T/\delta)} = N. \tag{8}$$

B. Proof of Theorem 1

First, we show the definition of the Neural Tangent Kernel (NTK) matrix and its effective dimension.

Definition 1. *Let $\boldsymbol{M}$ denote the NTK matrix associated with $\mathcal{X}_T$. The effective dimension $\tilde{d}$ of $\boldsymbol{M}$, with regularization parameter λ, is defined by: $\tilde{d} = \frac{\log\det(\boldsymbol{I}+\boldsymbol{M}/\lambda)}{\log(1+TN/\lambda)}$.*

The NTK matrix $\boldsymbol{M}$ is constructed recursively, layer by layer, from the input to the output of the neural network [50]. The effective dimension $\tilde{d}$ captures the intrinsic dimensionality of the input in the Reproducing Kernel Hilbert Space (RKHS) induced by the NTK. For additional details, we refer the reader to [29].

Lemma 3. (Lemma 5.2 [29]) *There exist positive constants $\bar{C}_1, \bar{C}_2$ such that for any $\delta \in (0,1)$, if $\eta \leq \bar{C}_1(TwL+w\lambda)^{-1}$ and $w \geq \bar{C}_2 \max\left\{T^7\lambda^{-7}L^{21}(\log w)^3, \lambda^{-\frac{1}{2}}L^{-\frac{3}{2}}(\log(TNL^2/\delta))^{\frac{3}{2}}\right\}$, then with probability at least $1-\delta$, we have: $\|\boldsymbol{\xi}_t - \boldsymbol{\xi}_0\|_2 \leq 2\sqrt{t/(w\lambda)}, \|\boldsymbol{\xi}^* - \boldsymbol{\xi}_t\|_{\boldsymbol{Z}_t} \leq \gamma_t/\sqrt{w}, \forall t \in \mathcal{T}$.*

Lemma 4. *Let (h_t^*, i_t^*) be the leaf node that contains $\boldsymbol{x}_t^*$. There exist positive constants $\overline{C}_1, \overline{C}_2$ such that for any $\delta \in (0,1)$, if η and w satisfy the same conditions as in Lemma 3, then with probability at least $1-\delta$, we have:*

$$f_1^*(\boldsymbol{x}_t^*) - f_1^*(\boldsymbol{x}_{h_t,i_t}) \leq 2\min\{\gamma_{t-1}\|\boldsymbol{g}(\boldsymbol{x}_{h_t,i_t}, \boldsymbol{\xi}_{t-1})/\sqrt{w}\|_{\boldsymbol{Z}_{t-1}}, 1\} + 2\bar{\Gamma} + \nu_1\rho^{h_t}, \tag{9}$$

where $\bar{\Gamma} = \overline{C}_1 w^{-1/6}\sqrt{\log w} t^{2/3}\lambda^{-2/3}L^3 + \overline{C}_2(\sqrt{2\boldsymbol{m}^\top \boldsymbol{M}^{-1}\boldsymbol{m}} + 2\sqrt{t/\lambda})w^{-1/6}\sqrt{\log w}t^{1/6}\lambda^{-1/6}L^{7/2}$.

We now prove that the cumulative regret satisfies:

$$\begin{aligned}
R_T =& \sum_{t\in\mathcal{T}} [f_1^*(\boldsymbol{x}_t^*) - f_1^*(\boldsymbol{x}_{h_t,i_t})] \\
\leq& 2\sqrt{T\sum_{t\in\mathcal{T}} \min\{\gamma_{t-1}^2\|\boldsymbol{g}(\boldsymbol{x}_{h_t,i_t}, \boldsymbol{\xi}_{t-1})/\sqrt{w}\|_{\boldsymbol{Z}_{t-1}}^2, 1\}} \\
&+ 2T\bar{\Gamma} + \sum_{t\in\mathcal{T}} \nu_1\rho^{h_t} \\
\leq& 3\sqrt{T}\sqrt{\tilde{d}\log(1+TN/\lambda) + 2}\cdot \\
&\Big[v\sqrt{\tilde{d}\log(1+TN/\lambda) + 2 - 2\log\delta} \\
&+(\lambda + C_3TL)(1-\eta w\lambda)^{J/2}\sqrt{T/\lambda} + 2\sqrt{\lambda S}\Big] + T + 1.
\end{aligned} \tag{10}$$

where the first inequality follows from Lemma 4 and the Cauchy–Schwarz inequality; the second inequality follows by applying Lemma 5.4 of [29], valid for sufficiently large w. This completes the proof of Theorem 1.

C. Proof of Lemma 4

For the difference between the estimated value and the real value, we have:

$$\begin{aligned}
&\|f(\boldsymbol{x}, \boldsymbol{\xi}_{t-1}) - < \boldsymbol{g}(\boldsymbol{x}, \boldsymbol{\xi}_0), \boldsymbol{\xi}^* - \boldsymbol{\xi}_0 > \| \\
=&\|f(\boldsymbol{x}, \boldsymbol{\xi}_{t-1}) - f(\boldsymbol{x}, \boldsymbol{\xi}_0) - < \boldsymbol{g}(\boldsymbol{x}, \boldsymbol{\xi}_0), \boldsymbol{\xi}_{t-1} - \boldsymbol{\xi}_0 > \\
&- < \boldsymbol{g}(\boldsymbol{x}, \boldsymbol{\xi}_0), \boldsymbol{\xi}^* - \boldsymbol{\xi}_{t-1} > \| \\
\leq& \overline{C}_1 w^{-1/6}\sqrt{\log w} t^{2/3}\lambda^{-2/3}L^3 + \| < \boldsymbol{g}(\boldsymbol{x}, \boldsymbol{\xi}_0), \boldsymbol{\xi}^* - \boldsymbol{\xi}_{t-1} > \|,
\end{aligned} \tag{11}$$

where the first equality holds due to $f(\boldsymbol{x}, \boldsymbol{\xi}_0)$ by the random initialization of $\boldsymbol{\xi}_0$, the inequality holds due to Lemma 4.1 of [51] with the fact $\|\boldsymbol{\xi}_{t-1} - \boldsymbol{\xi}_0\|_2 \leq 2\sqrt{t/(w\lambda)}$. For the second term, we have the following bound:

$$\begin{aligned}
&\| < \boldsymbol{g}(\boldsymbol{x}, \boldsymbol{\xi}_0), \boldsymbol{\xi}^* - \boldsymbol{\xi}_{t-1} > \| \\
\leq& \| < \boldsymbol{g}(\boldsymbol{x}, \boldsymbol{\xi}_{t-1}), \boldsymbol{\xi}^* - \boldsymbol{\xi}_{t-1} > \| + \\
&\| < \boldsymbol{g}(\boldsymbol{x}, \boldsymbol{\xi}_0) - \boldsymbol{g}(\boldsymbol{x}, \boldsymbol{\xi}_{t-1}), \boldsymbol{\xi}^* - \boldsymbol{\xi}_{t-1} > \| \\
\leq& \|\boldsymbol{\xi}^* - \boldsymbol{\xi}_{t-1}\|_{\boldsymbol{Z}_{t-1}} \|\boldsymbol{g}(\boldsymbol{x}, \boldsymbol{\xi}_{t-1})\|_{\boldsymbol{Z}_{t-1}} + \\
&\|\boldsymbol{\xi}^* - \boldsymbol{\xi}_{t-1}\|_2 \|\boldsymbol{g}(\boldsymbol{x}, \boldsymbol{\xi}_0) - \boldsymbol{g}(\boldsymbol{x}, \boldsymbol{\xi}_{t-1})\|_2 \\
\leq& \gamma_{t-1}\|\boldsymbol{g}(\boldsymbol{x}, \boldsymbol{\xi}_{t-1})/\sqrt{w}\|_{\boldsymbol{Z}_{t-1}} + \\
&\|(\boldsymbol{\xi}^* - \boldsymbol{\xi}_0) + (\boldsymbol{\xi}_0 - \boldsymbol{\xi}_{t-1})\|_2 \|\boldsymbol{g}(\boldsymbol{x}, \boldsymbol{\xi}_0) - \boldsymbol{g}(\boldsymbol{x}, \boldsymbol{\xi}_{t-1})\|_2 \\
\leq& \gamma_{t-1}\|\boldsymbol{g}(\boldsymbol{x}, \boldsymbol{\xi}_{t-1})/\sqrt{w}\|_{\boldsymbol{Z}_{t-1}} + \\
&\overline{C}_2(\sqrt{2\boldsymbol{m}^\top \boldsymbol{M}^{-1}\boldsymbol{m}} + 2\sqrt{t/\lambda})w^{-1/6}\sqrt{\log w}t^{1/6}\lambda^{-1/6}L^{7/2},
\end{aligned} \tag{12}$$

where the first inequality is derived from the triangle inequality; the second inequality holds due to Hölder's inequality; the third inequality follows Lemma 3; the fourth inequality uses Lemmas B.5 and B.6 in [29]. Combining Eq. (11) and Eq. (12), we obtain:

$$\begin{aligned}
&f(\boldsymbol{x}, \boldsymbol{\xi}_{t-1}) - \langle \boldsymbol{g}(\boldsymbol{x}, \boldsymbol{\xi}_0), \boldsymbol{\xi}^* - \boldsymbol{\xi}_0 \rangle\| \leq \\
&\gamma_{t-1}\|\boldsymbol{g}(\boldsymbol{x}, \boldsymbol{\xi}_{t-1})/\sqrt{w}\|_{\boldsymbol{Z}_{t-1}} + \bar{\Gamma}.
\end{aligned} \tag{13}$$

Now, we can compute:

$$\begin{aligned}
&f_1^*(\boldsymbol{x}_t^*) - f_1^*(\boldsymbol{x}_{h_t,i_t}) \\
=& f_1^*(\boldsymbol{x}_t^*) - f_1^*(\boldsymbol{x}_{h_t^*,i_t^*}) + f_1^*(\boldsymbol{x}_{h_t^*,i_t^*}) - f(\boldsymbol{x}_{h_t^*,i_t^*}, \boldsymbol{\xi}_{t-1}) + \\
&f(\boldsymbol{x}_{h_t^*,i_t^*}, \boldsymbol{\xi}_{t-1}) - f(\boldsymbol{x}_{h_t,i_t}, \boldsymbol{\xi}_{t-1}) + f(\boldsymbol{x}_{h_t,i_t}, \boldsymbol{\xi}_{t-1}) - f_1^*(\boldsymbol{x}_{h_t,i_t}) \\
\leq& \|f_1^*(\boldsymbol{x}_t^*) - f_1^*(\boldsymbol{x}_{h_t^*,i_t^*})\| + \|f_1^*(\boldsymbol{x}_{h_t^*,i_t^*}) - f(\boldsymbol{x}_{h_t^*,i_t^*}, \boldsymbol{\xi}_{t-1})\| + \\
&f(\boldsymbol{x}_{h_t^*,i_t^*}, \boldsymbol{\xi}_{t-1}) - f(\boldsymbol{x}_{h_t,i_t}, \boldsymbol{\xi}_{t-1}) + \|f(\boldsymbol{x}_{h_t,i_t}, \boldsymbol{\xi}_{t-1}) - f_1^*(\boldsymbol{x}_{h_t,i_t})\| \\
\leq& 2\gamma_{t-1}\|\boldsymbol{g}(\boldsymbol{x}_{h_t,i_t}, \boldsymbol{\xi}_{t-1})/\sqrt{w}\|_{\boldsymbol{Z}_{t-1}} + 2\bar{\Gamma} + \nu_1\rho^{h_t} \\
\leq& \min\{2\gamma_{t-1}\|\boldsymbol{g}(\boldsymbol{x}_{h_t,i_t}, \boldsymbol{\xi}_{t-1})/\sqrt{w}\|_{\boldsymbol{Z}_{t-1}} + 2\bar{\Gamma} + \nu_1\rho^{h_t}, 1\} \\
\leq& 2\min\{\gamma_{t-1}\|\boldsymbol{g}(\boldsymbol{x}_{h_t,i_t}, \boldsymbol{\xi}_{t-1})/\sqrt{w}\|_{\boldsymbol{Z}_{t-1}}, 1\} + 2\bar{\Gamma} + \nu_1\rho^{h_t},
\end{aligned} \tag{14}$$

where the first inequality follows from the triangle inequality and the fact that the difference between any two scalar values can be upper bounded by the sum of their pairwise differences; the second inequality applies the local smoothness assumption and the bound on the prediction error, as given in Eq. (13); the third inequality holds because the reward function is bounded within $[0,1]$, which can be achieved through normalization; the fourth inequality follows from the property that $\min\{a+b, 1\} \leq \min\{a,1\} + b$ for any non-negative values a and b.

REFERENCES

[1] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," in *Proc. of NAACL*, 2019, pp. 4171–4186.

[2] C. Raffel, N. Shazeer, A. Roberts, K. Lee, S. Narang, M. Matena, Y. Zhou, W. Li, and P. J. Liu, "Exploring the limits of transfer learning with a unified text-to-text transformer," *Journal of machine learning research*, vol. 21, no. 140, pp. 1–67, 2020.

[3] X. Dai, Y. Xie, M. Liu, X. Wang, Z. Li, H. Wang, and J. Lui, "Multi-agent conversational online learning for adaptive llm response identification," *arXiv preprint arXiv:2501.01849*, 2025.

[4] H. Huang, N. Ardalani, A. Sun, L. Ke, H.-H. S. Lee, S. Bhosale, C.-J. Wu, and B. Lee, "Toward efficient inference for mixture of experts," in *Proc. of NeurIPS*, vol. 37, 2024, pp. 84 033–84 059.

[5] W. Fedus, B. Zoph, and N. Shazeer, "Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity," *Journal of Machine Learning Research*, vol. 23, no. 120, pp. 1–39, 2022.

[6] M. R. Costa-Jussà, J. Cross, O. Çelebi, M. Elbayad, K. Heafield, K. Heffernan, E. Kalbassi, J. Lam, D. Licht, J. Maillard *et al.*, "No language left behind: Scaling human-centered machine translation," *arXiv preprint arXiv:2207.04672*, 2022.

[7] Y. Huang, P. Ye, X. Huang, S. Li, T. Chen, T. He, and W. Ouyang, "Experts weights averaging: A new general training scheme for vision transformers," *arXiv preprint arXiv:2308.06093*, 2023.

[8] M. Liu, W. Wang, and C. Wu, "Optimizing distributed deployment of mixture-of-experts model inference in serverless computing," in *Proc. of IEEE INFOCOM*, 2025, pp. 1–10.

[9] L. Jin, Y. Zhang, Y. Li, S. Wang, H. H. Yang, J. Wu, and M. Zhang, "Moe^2: Optimizing collaborative inference for edge large language models," *arXiv preprint arXiv:2501.09410*, 2025.

[10] S. Kudugunta, Y. Huang, A. Bapna, M. Krikun, D. Lepikhin, M.-T. Luong, and O. Firat, "Beyond distillation: Task-level mixture-of-experts for efficient inference," *arXiv preprint arXiv:2110.03742*, 2021.

[11] M. Muqeeth, H. Liu, and C. Raffel, "Soft merging of experts with adaptive routing," *arXiv preprint arXiv:2306.03745*, 2023.

[12] S. He, R.-Z. Fan, L. Ding, L. Shen, T. Zhou, and D. Tao, "Merging experts into one: Improving computational efficiency of mixture of experts," *arXiv preprint arXiv:2310.09832*, 2023.

[13] Z. Zhong, M. Xia, D. Chen, and M. Lewis, "Lory: Fully differentiable mixture-of-experts for autoregressive language model pre-training," *arXiv preprint arXiv:2405.03133*, 2024.

[14] P. Li, Z. Zhang, P. Yadav, Y.-L. Sung, Y. Cheng, M. Bansal, and T. Chen, "Merge, then compress: Demystify efficient SMoe with hints from its routing policy," in *Proc. of ICLR*, 2024.

[15] H. Li and L. Duan, "Theory of mixture-of-experts for mobile edge computing," in *Prof. of NFOCOM*, 2025, pp. 1–10.

[16] G. Cormode and S. Muthukrishnan, "An improved data stream summary: the count-min sketch and its applications," *Journal of Algorithms*, vol. 55, no. 1, pp. 58–75, 2005.

[17] N. Shazeer, A. Mirhoseini, K. Maziarz, A. Davis, Q. Le, G. Hinton, and J. Dean, "Outrageously large neural networks: The sparsely-gated mixture-of-experts layer," *arXiv preprint arXiv:1701.06538*, 2017.

[18] J. Liu, J. H. Wang, and Y. Jiang, "Janus: A unified distributed training framework for sparse mixture-of-experts models," in *Proc. of the ACM SIGCOMM*, 2023, pp. 486–498.

[19] S. Shi, X. Pan, X. Chu, and B. Li, "Pipemoe: Accelerating mixture-of-experts through adaptive pipelining," in *Proc. of IEEE INFOCOM*, 2023, pp. 1–10.

[20] N. Wang, W. Lin, L. Zhang, S. Shi, R. Zhou, and B. Li, "Sp-moe: Expediting mixture-of-experts training with optimized pipelining planning," in *Proc. of IEEE INFOCOM*, 2025, pp. 1–10.

[21] E. Yang, L. Shen, G. Guo, X. Wang, X. Cao, J. Zhang, and D. Tao, "Model merging in llms, mllms, and beyond: Methods, theories, applications and opportunities," *arXiv preprint arXiv:2408.07666*, 2024.

[22] G. Ilharco, M. T. Ribeiro, M. Wortsman, L. Schmidt, H. Hajishirzi, and A. Farhadi, "Editing models with task arithmetic," in *Proc. of ICLR*, 2023.

[23] E. Yang, Z. Wang, L. Shen, S. Liu, G. Guo, X. Wang, and D. Tao, "Adamerging: Adaptive model merging for multi-task learning," in *Proc. of ICLR*, 2024.

[24] X. Jin, X. Ren, D. Preotiuc-Pietro, and P. Cheng, "Dataless knowledge fusion by merging weights of language models," in *Proc. of ICLR*, 2023.

[25] M. S. Matena and C. A. Raffel, "Merging models with fisher-weighted averaging," in *Proc. of NeurIPS*, vol. 35, 2022, pp. 17 703–17 716.

[26] T. D K, G. Nathan, and S. M S, "Fisher mask nodes for language model merging," in *Proc. of LREC-COLING*, May 2024, pp. 7349–7355.

[27] P. Xu, Z. Wen, H. Zhao, and Q. Gu, "Neural contextual bandits with deep representation and shallow exploration," in *Proc. of ICLR*, 2022.

[28] Y. Qi, Y. Ban, A. Banerjee, and J. He, "Robust neural contextual bandit against adversarial corruptions," *Proc. of NeurIPS*, vol. 37, pp. 19 378–19 446, 2024.

[29] D. Zhou, L. Li, and Q. Gu, "Neural contextual bandits with UCB-based exploration," in *Proc. of ICML*, vol. 119, 13–18 Jul 2020, pp. 11 492–11 502.

[30] W. ZHANG, D. Zhou, L. Li, and Q. Gu, "Neural thompson sampling," in *Proc. of ICLR*, 2021.

[31] Y. Ban, Y. Yan, A. Banerjee, and J. He, "EE-net: Exploitation-exploration neural networks in contextual bandits," in *Proc. of ICLR*, 2022.

[32] S. Wang, S. Bian, X. Liu, and Z. Shao, "Neural constrained combinatorial bandits," *IEEE Transactions on Networking*, 2025.

[33] R. A. Jacobs, M. I. Jordan, S. J. Nowlan, and G. E. Hinton, "Adaptive mixtures of local experts," *Neural computation*, vol. 3, no. 1, pp. 79–87, 1991.

[34] S. Sukhbaatar, O. Golovneva, V. Sharma, H. Xu, X. V. Lin, B. Roziere, J. Kahn, S.-W. Li, W. tau Yih, J. E. Weston, and X. Li, "Branch-train-mix: Mixing expert LLMs into a mixture-of-experts LLM," in *Proc. of COLM*, 2024.

[35] J. Gast and S. Roth, "Lightweight probabilistic deep networks," in *Proc. of CVPR*, 2018, pp. 3369–3378.

[36] Z. Han, R. Zhou, C. Xu, Y. Zeng, and R. Zhang, "Inss: An intelligent scheduling orchestrator for multi-gpu inference with spatio-temporal sharing," *IEEE Transactions on Parallel and Distributed Systems*, 2024.

[37] H. Kellerer, U. Pferschy, D. Pisinger, H. Kellerer, U. Pferschy, and D. Pisinger, *Multidimensional knapsack problems*. Springer, 2004.

[38] J. Pang, Z. Han, R. Zhou, R. Zhang, J. C. Lui, and H. Chen, "Eris: An online auction for scheduling unbiased distributed learning over edge networks," *IEEE Transactions on Mobile Computing*, vol. 23, no. 6, pp. 7196–7209, 2023.

[39] Y. Fu, X. Zhang, and M. Zhang, "Heterogeneous mean-field reinforcement learning for age-minimal gpu batching," in *Proc. of IEEE INFOCOM*, 2026.

[40] D. Bouneffouf and R. Féraud, "A tutorial on multi-armed bandit applications for large language models," in *Proc. of ACM SIGKDD*, 2024, pp. 6412–6413.

[41] X. Dai, J. Li, X. Liu, A. Yu, and J. Lui, "Cost-effective online multi-llm selection with versatile reward models," *arXiv preprint arXiv:2405.16587*, 2024.

[42] A. Lazaric, E. Brunskill *et al.*, "Online stochastic optimization under correlated bandit feedback," in *Proc. of ICML*, 2014, pp. 1557–1565.

[43] G. B. Dantzig, "Linear programming and extensions," 2016.

[44] W. Chu, L. Li, L. Reyzin, and R. Schapire, "Contextual bandits with linear payoff functions," in *Proc. of AISTATS*, 2011, pp. 208–214.

[45] A. Wang, A. Singh, J. Michael, F. Hill, O. Levy, and S. R. Bowman, "Glue: A multi-task benchmark and analysis platform for natural language understanding," *arXiv preprint arXiv:1804.07461*, 2018.

[46] Z. Han, H. Wang, Z. Zhang, X. Dai, X. Liu, and J. Lui, "Hilora: Adaptive hierarchical lora routing for training-free domain generalization," *arXiv preprint arXiv:2510.12266*, 2025.

[47] A. Clark, D. de Las Casas, A. Guy, A. Mensch, M. Paganini, J. Hoffmann, B. Damoc, B. Hechtman, T. Cai, S. Borgeaud *et al.*, "Unified scaling laws for routed language models," in *Proc. of ICML*, 2022, pp. 4057–4086.

[48] S. P. Singh and M. Jaggi, "Model fusion via optimal transport," *Proc. of NeurIPS*, vol. 33, pp. 22 045–22 055, 2020.

[49] B. Atalar and C. Joe-Wong, "Neural combinatorial clustered bandits for recommendation systems," in *Proc. of the AAAI*, vol. 39, no. 15, 2025, pp. 15 417–15 426.

[50] A. Jacot, F. Gabriel, and C. Hongler, "Neural tangent kernel: Convergence and generalization in neural networks," *Proc. of NeurIPS*, vol. 31, 2018.

[51] Y. Cao and Q. Gu, "Generalization bounds of stochastic gradient descent for wide and deep neural networks," in *Proc. of NeurIPS*, vol. 32, 2019.