sMoE: Elastic MoE-Based Inference with Serverless Computing via Fine-grained Expert Scaling

Xiaofei Yue[1,2], Ziming Zhao[1*], Jiongchi Yu[3], Huidong Ma[4], Zhaoxuan Li[5], Tingting Li[1*], Jianwei Yin[1*]

[1]School of Software Technology, Zhejiang University, Ningbo, China. [2]Beijing Institute of Technology, Beijing, China.
[3]Singapore Management University, Singapore. [4]Nankai University, Tianjin, China. [5]IIE, CAS, Beijing, China.

Abstract

Mixture-of-Experts (MoE) architecture enables large-scale inference via sparse activation, but skewed, time-varying expert popularity leads to cost-inefficiency. Serverless computing is an attractive substrate due to its fine-grained resource elasticity and billing. Even so, achieving cost-efficient and SLO-compliant scaling for each expert with intra- and inter-layer dependencies under concurrent requests remains fraught with challenges. In this paper, we present sMoE, a topology-aware elastic auto-scaler for serverless MoE inference. Our insight is to treat the deployment of a MoE inference pipeline as a DAG of experts and non-MoE segments. Building on this, sMoE is designed as a deep reinforcement learning-based solution. Specifically, it encodes expert- and layer-level runtime features with DAGNN by propagating cross-layer semantics. Coupled with a layer-wise pointer network, sMoE captures intra-layer semantics to jointly decide vertical, horizontal configurations, and concurrency setting for each expert at runtime. Experimental results show that sMoE reduces serving cost by 21.4%–39.2% compared to state-of-the-art solutions, while meeting stringent P95 latency SLOs.

ACM Reference Format:
Xiaofei Yue, Ziming Zhao, Jiongchi Yu, Huidong Ma, Zhaoxuan Li, Tingting Li, and Jianwei Yin. 2026. sMoE: Elastic MoE-Based Inference with Serverless Computing via Fine-grained Expert Scaling. In *63rd ACM/IEEE Design Automation Conference (DAC '26), July 26–29, 2026, Long Beach, CA, USA.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3770743.3804029

1 Introduction

Large models are increasingly deployed as online services to serve applications such as content understanding [1] and conversational AI [2]. To keep inference affordable at scale, Mixture-of-Experts (MoE) architectures [3] are widely-used by replacing Transformer blocks as MoE layers, as shown in Figure 1. Concretely, each token is routed to a subset of experts by the gating network for aggregation, trading dense computation with sparse activation [4]. Thus, some experts within a layer might serve fewer payloads (*i.e.,* tokens) and finish earlier, leaving them to remain idle for others, as shown in Figure 1(b). This *skewed expert popularity* [5] slows down inference (hot experts), and amplifies costs due to the idle, cold experts.

Serverless computing has emerged as an appealing substrate for AI inference [6, 7], offering auto-scaling, pay-per-use billing, and simplified operations. Building on this, each expert can be hosted as a lightweight function with fine-grained resources, and organized into a pipeline [8]. Tokens are easily passed between functions with requests [9]. Moreover, the platform handles underlying provisioning. Figure 1(c) displays that serverless functions can naturally mitigate the resource idling issues of MoE layers. Nevertheless, ensuring Service Level Objectives (SLO) for latency-sensitive inference services and enhancing resource efficiency remain basic demands for cloud vendors and users. Existing works have explored resource management for individual inference functions [10–12].

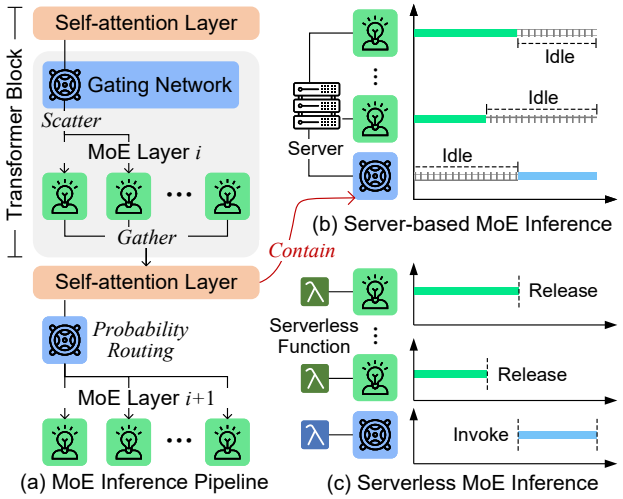


Figure 1: The overview of a MoE inference pipeline.

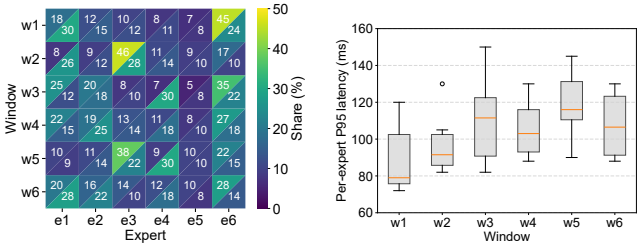
Despite their success, we observe that serverless MoE inference exhibits unique characteristics and ensuing challenges. ***C1: Trade-off between SLO and cost under concurrent requests.*** The multi-layer, multi-expert structure of an MoE pipeline prolongs the end-to-end inference path. This poses greater challenges in ensuring SLO compared to individual functions, especially with concurrent requests. Blindly scaling experts could easily lead to significant resource and cost wastage. ***C2: Intricate inter-expert dependencies.*** Experts are coupled due to *scatter-gather operations* [4], creating intricate intra- and inter-layer dependencies throughout the MoE pipeline, which leads to cascading performance effects [13]. Thus, the configuration space grows exponentially with both the number of (*i*) MoE layers and (*ii*) experts. ***C3: Time-varying, skewed expert popularity.*** Expert popularity within MoE layers varies across requests due to input distribution drift or shifting request patterns [14, 15]. This leads to dynamic token and request imbalance, along with the inter-expert dependencies, making it tough to capture and mitigate straggler behavior accurately.

Existing works are ill-suited to this setting. Static optimization derives an expert layout (*e.g.,* memory limits, and replica sizing) for

*Corresponding authors: {zhaoziming, litt2020, zjuyjw}@zju.edu.cn

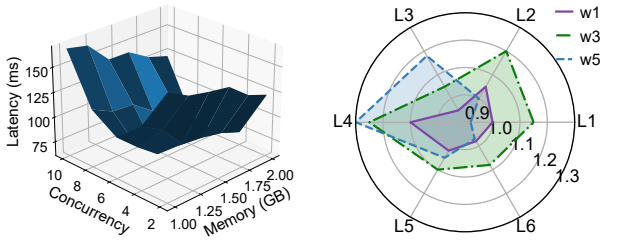
DAC '26, Long Beach, CA, USA

ACM ISBN 979-8-4007-2254-7/2026/07
https://doi.org/10.1145/3770743.3804029



(a) Token/Concurrency share. (b) Tail latency.

Figure 2: The payload and performance skew of MoE layers.



(a) Expert configuration space. (b) Performance changes.

Figure 3: The performance and dependencies of MoE layers.

a fixed workload profile [8, 9]. They are re-calibrated as the expert popularity drifts, which is costly at scale and abstracts away runtime effects (*e.g.,* bursty requests [7, 14]). General-purpose autoscaling (*e.g.,* QPS-driven scaling) treats each function in isolation [10, 11], and is oblivious to the topology and scatter-gather coupling of pipelines, optimizing local utilization instead of end-to-end SLOs.

To fill these gaps, we propose sMoE, a topology-aware elastic auto-scaler for serverless MoE inference. Our key observation is that expert stragglers are jointly shaped by horizontal and vertical resources, token payloads, and concurrency. Accordingly, sMoE adopts a scalable *GNN-based Pointer Reinforcement Learning (GPRL)* algorithm to derive fine-grained expert-level scaling decisions for cost-efficient configuration and SLO compliance (**C1**). Treating the MoE pipeline as a DAG, sMoE uses a DAGNN for MoE-layer decomposition by aggregating their partial-order token dependencies into expert embedding (**C2**). Also, sMoE designs a layer-wise pointer-network actor to facilitate batch decision-making, while capturing intra-layer expert context to mitigate stragglers (**C2** and **C3**).

In summary, this paper makes the following contributions:

- We formulate the elastic, fine-grained expert scaling problem for serverless MoE inference, whose goal is to reduce resource costs and ensure SLOs under concurrent requests.
- We propose sMoE, a GPRL-based auto-scaler that combines a DAGNN with a layer-wise pointer network to decide horizontal and vertical configurations for each expert.
- We implement a prototype of sMoE, which outperforms the state-of-the-art solutions in serving cost and SLO compliance via extensive experiments.

2 Background and Motivation

2.1 Serverless MoE Inference

Mixture-of-Experts (MoE) architectures extend Transformer models with sparse expert layers, where a router produces expert scores via a gating network and routes each token to the top-k experts, whose outputs are then aggregated [4, 5]. By increasing the number of experts while activating only a small subset per token, MoE scales model capacity with reduced per-request compute.

Serverless platforms (*e.g.,* OpenFaaS [16]) offer an appealing substrate for MoE models. Each expert and non-MoE segment (*e.g.,* gating network and self-attention layer in Figure 1) can be packaged as a function with configurable resources (*e.g.,* memory, CPU and GPU). The platform manages instance provisioning, request routing, and pay-per-use billing. This yields a pipeline-style MoE inference service, where requests traverse alternating these functions. However, each MoE layer exhibits a *scatter-gather* pattern: tokens are routed to the top-k experts for parallel processing, and the layer completes only after all routed experts return, thus forming a synchronization barrier. That is, MoE-layer latency is dominated by the slowest expert (*i.e.,* a straggler) and can be further amplified as requests propagate via the pipeline, while resulting in underutilization of other experts.

2.2 Motivation

Observation 1: Apart from tokens, the number of concurrent requests among experts is also skewed. We first deploy a representative MoE model on OpenFaaS and replay a non-stationary request trace (see § 5.1). For each time window, we measure the shares of token and concurrent requests for each expert with a MoE layer, as shown in Figure 2(a). The left (right) triangle encodes the token (concurrency) share. Across windows, both token and concurrency distributions are highly skewed and drift over time. More importantly, they are *not symmetric*: some experts receive only a moderate fraction of tokens but accumulate a disproportionately large fraction of requests due to slower processing. Figure 2(b) shows that these experts become clear tail-latency hotspots, indicating that the performance of a MoE layer is jointly hindered by token skew and concurrency skew.

Observation 2: Concurrency setting is as critical as hardware resources. We then conduct a micro-benchmark on an expert by sweeping its memory limit (which proportionally configures CPU/GPU share in OpenFaaS) and concurrency setting. As shown in Figure 3(a), increasing memory generally reduces latency, and moderate concurrency improves utilization and amortizes overheads. However, overly high concurrency causes queueing and sharply increases tail latency. The surface is clearly non-monotonic along the concurrency dimension and exhibits "ridges" and "valleys" where certain memory-concurrency combinations are markedly better than nearby settings. This confirms that expert performance cannot be controlled by tuning vertical resources alone and concurrency must be co-optimized to avoid underutilization at low concurrency and severe stragglers at high concurrency.

Observation 3: Expert dependencies are heterogeneous and time-varying. Finally, we examine how bottlenecks shift across MoE layers over time. Figure 3(b) shows the normalized P95 latency of each MoE layer. Early layers dominate the latency budget in some windows, whereas in others the bottlenecks migrate to middle or deeper layers. This drifting pattern implies that cross-layer dependencies and critical paths in the MoE pipeline are dynamic. A static configuration or per-function heuristic that ignores the

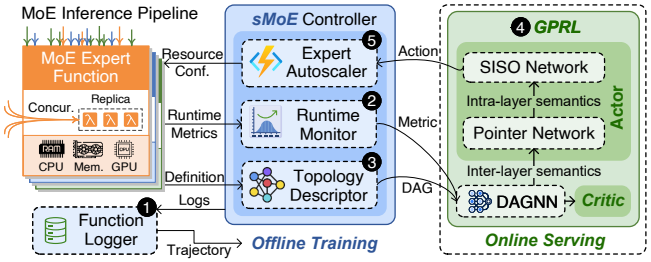


Figure 4: The architecture overview of sMoE.

pipeline structure and its temporal evolution will either miss emerging hotspots or overprovision layers that are no longer critical.

3 System Overview

System Model. We consider a serverless MoE inference service G with a well-trained model consisting of n MoE layers $\mathcal{L} = \{\ell_1, \ell_2, ..., \ell_n\}$. Each MoE layer ℓ_i hosts multiple experts $\mathcal{E}_i = \{e_{i,1}, \ldots, e_{i,K_i}\}$, where K_i is the number of experts in layer ℓ_i. Each expert is mapped to a unique function for Expert Parallelism (EP) [8]. As widely studied in prior work [17, 18], the $n+1$ non-MoE parts (including gating networks) $\mathcal{P} = \{p_1, ..., p_{n+1}\}$ between MoE layers are split and pipelined for model parallelism. Coupled with the token skew and synchronization bottlenecks of MoE layers (see §2), we focus on optimizing the EP in this paper. sMoE provides each expert $e_{i,j}$ with a holistic, hybrid configuration $\phi = \{\phi_{mem}, \phi_{cpu}, \phi_{gpu}, \phi_{rep}, \phi_{ccr}\} \in \Phi$ under heterogeneous hardware resources, including the memory limits, GPU share, numbers of CPU cores, function instances (*i.e.,* replicas), and concurrency, respectively.

Elastic MoE Inference Scaling. We consider a series of timesteps $\mathcal{T}$, each of which is divided evenly. The concurrent inference requests $R(t)$ will arrive continuously at any time in $t \in \mathcal{T}$. At each timestep t, we need to decide the elastic scaling $x^{(t)}_{i,j,\phi}$ (*i.e.,* a variable that indicates whether configuration ϕ is selected) for each expert $e_{i,j}$. The goal is to minimize the overall resource cost of requests $R(t)$ while ensuring they satisfy the end-to-end SLO. Thus, the *elastic MoE inference scaling problem* is formulated as:

$$\min_{X} \quad \sum_{t\in\mathcal{T}} C_{R(t)} \tag{1}$$

$$\text{s.t.} \quad \sum_{\phi\in\Phi} x^{(t)}_{i,j,\phi}\left(\alpha\phi_{mem} - \frac{2D\cdot P^{(t)}_{i,j}}{\phi_{rep}}\right) \geq M_{e,i},\ \forall \ell_i, e_{i,j}, t \tag{2}$$

$$P95_{r\in R(t)}\,\tau_r \leq \lambda,\ \forall t \tag{3}$$

$$(\phi_{cpu} = 0) \oplus (\phi_{gpu} = 0) = 1,\ \forall \phi \tag{4}$$

$$\sum_{\phi\in\Phi} x^{(t)}_{i,j,\phi} = 1,\ \forall \ell_i, e_{i,j}, t \tag{5}$$

$$x^{(t)}_{i,j,\phi} \in \{0,1\},\ \forall \ell_i, e_{i,j}, \phi, t \tag{6}$$

The cost $C_{R(t)}$ of serving requests $R(t)$ concurrently in objective (1) is the aggregated cost of all expert instances, given by:

$$C_{R(t)} = \sum_{\ell_i\in\mathcal{L}}\sum_{e_{i,j}\in\mathcal{E}_i}\sum_{\phi\in\Phi} x^{(t)}_{i,j,\phi}\phi_{rep}\tau^{R(t)}_{i,j}(\phi)U_\phi, \tag{7}$$

where U_ϕ represents the unit execution cost under ϕ, which is hardware-dependent. $\tau^{R(t)}_{i,j}$ is the latency of expert $e_{i,j}$ for serving $R(t)$ concurrently. Inequality (2) ensures that each expert instance has sufficient memory with a loss factor α to cover its inherent runtime $M_{e,i}$ (*e.g.,* model parameters and library), input and output

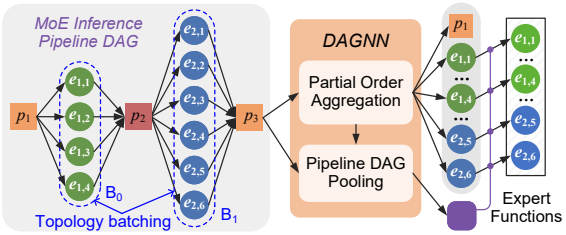


Figure 5: The embedding process of DAGNN in sMoE.

tokens for $e_{i,j}$, where D is the size of a token, and $P^{(t)}_{i,j}$ means the peak number of input tokens at t. Constraint (3) ensures the tail (*e.g.,* 95^{th}-percentile) latency over $R(t)$ meets SLO λ, where the inference latency τ_r of request r is defined as:

$$\tau_r = \sum_{\ell_i\in\mathcal{L}}\left(\max_{e_{i,j}\in\mathcal{E}_i}\sum_{\phi\in\Phi} x^{(t)}_{i,j,\phi}\,\tau^r_{i,j}(\phi) + T^{r,\text{NE}}_i\right) + T^{r,\text{NE}}_{n+1}, \tag{8}$$

where $\tau^r_{i,j}(\phi)$ and $T^{r,\text{NE}}_i$ are the latency of expert $e_{i,j}$ under ϕ and i-th non-MoE part for serving r, respectively. Equation (4) means an expert can be served using one type of computing resource. Constraints (5) and (6) refer to the domain constraints.

System Architecture. We show the architecture of sMoE in Figure 4. The insight of sMoE is to apply *GPRL* to learn an elastic, fine-grained expert scaling policy for serverless MoE inference pipelines under concurrent requests. Initially, we use historical data from ❶ Function Logger for Offline Training of the GPRL model. During the online serving, sMoE individually generates the optimal scaling policy for each expert based on its runtime states. The ❷ Runtime Monitor collects information at both the layer and expert levels, and intra-layer indicators of experts as the states of GPRL. The MoE pipeline topology is loaded by ❸ Topology Descriptor for identifying expert dependencies via GNN. Building on this, ❹ the agent in our GPRL can efficiently make elastic scaling actions for entire MoE layers, including vertical (*e.g.,* CPU, GPU and memory) and horizontal resources (*e.g.,* number of replicas), as well as concurrency for all of its experts. Finally, the actions are converted into new configurations by ❺ Expert Autoscaler.

4 System Design

4.1 Overview

In fact, scaling the MoE inference pipeline elastically over time for (1) is a *sequential decision-making* problem. Coupled with the non-uniform configuration across numerous experts, sMoE is designed as a scalable GPRL-based solution. This is because DRL has superior high-dimensional feature abstraction and decision-making capabilities for interactive inference environments. By doing so, we can explore the unknown expert performance (*i.e.,* popularity) $\tau^r_{i,j}$ and $\tau^{R(t)}_{i,j}$ by replaying experiences. Benefiting from our topology-aware state representation (§4.2) and pointer-driven action encoding (§4.3), sMoE uses only one agent to serve as an auto-scaler to manage all experts. With the state of each MoE layer at t as input, the agent outputs a series of actions to decide the configurations of its experts.

4.2 Topology-aware State Representation

As shown in Figure 5(left), we treat the serverless MoE inference pipeline as a DAG $G = (V, E)$, where $V = \{V_\text{E}, V_\text{NE}\}$. Each node $v \in V_\text{E}(V_\text{NE})$ refers to the function of an expert (a non-MoE segment),

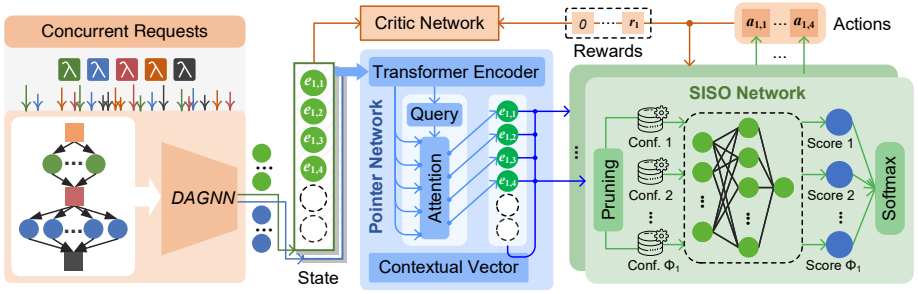


Figure 6: The model workflow of sMoE.

and edges E define their token dependencies. Naturally, the expert popularity of a MoE layer is shaped by cascading effects propagated from upstream layers, we desire to embed this inductive bias into the DAG representation. Aligning well with our goal, DAGNN [19] is driven by DAG-induced partial ordering. It can aggregate only the predecessor to get the embedding $h_v^{(l)}$ for each node v in DAG G at each layer l, and read out the graph embedding h_G via DAG pooling. We use the raw state space of an expert node $v : e_{i,j} \in V_E$ as its initial features $h_v^{(0)} \in \mathcal{H}_{\mathrm{E},t}^{(0)}$ using two-level metrics:

- ***At the expert level***, $\mathrm{RU}(t)$ covers the utilizations of hardware resources, and $\mathrm{CU}(t)$ is the concurrency utilization. The expert latency $\mathrm{EL}(t)$ is the tail latency when serving concurrent requests. The current configuration $\mathrm{CC}(t)$, tokens per second $\mathrm{TPS}(t)$, and requests per second $\mathrm{RPS}(t)$ are also part of the state.
- ***At the layer level***, $\mathrm{TS}(t)$ and $\mathrm{RS}(t)$ are the shares of tokens and requests routed to an expert within its layer, respectively. The token routing entropy $\mathrm{TRE}(t) = -\sum \mathrm{TS}(t) \log \mathrm{TS}(t)$ is smaller for more skewed popularity. $\mathrm{ELV}(t)$ is the variance of $\mathrm{EL}(t)$ among experts, and the latency straggler ratio $\mathrm{SR}(t)$ refers to the ratio of the tail latency of an expert to the straggler, serving for its lagging degree. Note that Floating-point Operations (FLOPS) $\mathrm{SF}(t)$ means the computational complexity of a MoE layer.

For a non-MoE segment node $v : p_i \in V_{NE}$, we treat it as a special expert node and use only the expert-level state as its initial features $h_v^{(0)} \in \mathcal{H}_{\mathrm{NE},t}^{(0)}$ for cross-layer semantic propagation among experts. Moreover, we adopt *Topological batching* to further node handling by dividing the DAG in topological order. Specifically, nodes without dependencies will join a sequential batch $B_i (i \geq 0)$. The experts of MoE layer l_i belong to the same batch B_{i-1} and can be handled concurrently, once the preceding non-MoE segment has completed. Then, we use the node features $\mathcal{H}_t = \{\mathcal{H}_{\mathrm{NE},t}, \mathcal{H}_{\mathrm{E},t}\}$ of pipeline DAG G as DAGNN's initial input $\mathcal{H}_t^{(0)}$:

$$\mathcal{H}_t^{(0)} = \left\{\mathcal{H}_{\mathrm{NE,t}}^{(0)}, \mathcal{H}_{\mathrm{E,t}}^{(0)}\right\} = \bigcup\nolimits_{v \in \mathcal{V}} h_v^{(0)}. \tag{9}$$

After L-layer message propagation with *topological batching*, we get a function embedding set $\mathcal{H}_t^{(L)} = \{\mathcal{H}_{\mathrm{NE,t}}^{(L)}, \mathcal{H}_{\mathrm{E,t}}^{(L)}\}$ and a pipeline embedding $h_{G,t}$. As shown in Figure 5, we concatenate the embedding $h_v^{(L)} \in \mathcal{H}_{\mathrm{E,t}}^{(L)}$ of each expert node v with $h_{G,t}$, forming its final DAGNN representations, $(h_v^{(L)}, h_{G,t})_{v:e_{i,j}} \mapsto h_{i,j,t}$, as the input state of the agent. By propagating and aggregating cross-layer token dependencies, we can separately scale each MoE layer. Next, we show how the agent encodes the scaling actions for a layer.

4.3 Pointer-driven Action Encoding

Pointer Module. To relieve stragglers caused by skewed expert popularity, experts within an MoE layer provide context for the scaling criticality of each other. Thus, the agent needs to process the per-expert state sequence $\mathcal{H}_{i,t} = \{h_{i,j,t} \mid e_{i,j} \in \ell_i\}$ for any MoE layer ℓ_i for intra-layer contextual semantics. Even within a single pipeline, however, the number of experts K_i may differ [20], preventing the learning from generalizing across layers. A direct method is to pad $\mathcal{H}_{i,t}$ to a fixed maximum length and adopt classical Seq2Seq models with positional encoding (e.g., Transformer [21]). However, the resulting output sequence is still anonymous and not explicitly aligned with expert identities, complicating the fine-grained scaling. In contrast, the pointer network [22] can learn a conditional distribution over input positions. As shown in Figure 6, we instantiate it with a Transformer encoder and an additive-attention decoder,

$$u_{i,j,t} = \begin{cases} -\infty, & \text{if MASK}\left(e_{i,j}\right) = 0, \\ v^T \tanh(W_1 \hat{h}_{i,j,t} + W_2 \hat{h}_{i,t}), & \text{o/w}, \end{cases} \tag{10}$$

where W_1, W_2, and v^T are learnable parameters, $\hat{h}_{i,j,t}$ is the refined Transformer embedding of $h_{i,j,t} \in \mathcal{H}_{i,t}$, which discovers intra-MoE layer ℓ_i correlations among experts for $e_{i,j}$, and $\hat{h}_{i,t} = \mathrm{avg}_{e_{i,j} \in \ell_i} \hat{h}_{i,j,t}$ serves as layer-wise summary. The attention score $u_{i,j,t}$ is used as a pointer to $e_{i,j}$, regarded as its potential scaling criticality under the current workload. Thus, we get the contextual vector $c_{i,t}$ of ℓ_i:

$$c_{i,t} = \sum\nolimits_{e_{i,j} \in \ell_i} \mathrm{Softmax}(u_{i,j,t}) \cdot \hat{h}_{i,j,t}, \tag{11}$$

where $u_{i,j,t}$ is converted into the weight of expert $e_{i,j}$ via a Softmax operator [22], since we should scale more resources to more critical experts to avoid intra-layer latency skew. Note that the score of the *padding* is masked to $-\infty$ and has a weight of 0. Finally, we build the further state $(\hat{h}_{i,j,t}, c_{i,t}) \mapsto s_{i,j,t}$ of expert $e_{i,j}$ for action encoding. By embedding explicit intra-layer dependencies, the scope of scaling decision-making is refined from layers to experts.

Action Filtering and Encoding. In contrast to imposing penalty terms for trial-and-error, we must filter out configurations that violate the memory constraint (2) before action encoding. This is because invalid actions may result in expert functions crashing directly at runtime [23]. To encode such a customized action space for each expert, sMoE defines the actor network as a Single-Input Single-Output (SISO) neural network, as shown in Figure 6 (right). Each state $s_{i,j,t}$ of expert $e_{i,j}$ concatenates with a total of $|\Phi|_i$ valid configurations to form the state set $S_{i,j,t} = (s_{i,j,t}^1, \ldots, s_{i,j,t}^{\phi}, \ldots, s_{i,j,t}^{|\Phi|_i})$, where $s_{i,j,t}^{\phi}$ refers to the state to select configuration ϕ for $e_{i,j}$. We feed $s_{i,j,t}^{\phi}$ into the SISO network to get a scalar value $g_{i,j,t}^{\phi}$, which

can be interpreted as the score of configuration ϕ [24]. Note that the SISO network can be served (*i.e.*, $|\Phi|_i$ tasks above) in parallel due to its structural design. Finally, we use the Softmax to transform these scores over valid configurations as our expert scaling policy:

$$x_{i,j,\phi}^{(t)} = \begin{cases} 1, & \phi = \arg\max_{\phi^*} a_{i,j,t}^{\phi^*}, \\ 0, & \text{otherwise}, \end{cases} \tag{12}$$

where $a_{i,j,t}^{\phi} = \text{Softmax}(g_{i,j,t}^{\phi})$.

4.4 Model Training

Reward Function. At timestep t, the agent receives an immediate reward 0 after it takes an action to scale expert $e_{i,j}$. The time in GPRL next moves forward, but the environment states do not change, until sMoE finishes the scaling of the last expert e_{i,K_i}. We now feed the agent with a reward for MoE layer ℓ_i, given by $r_{i,t} = -\beta \cdot C_{i,R(t)}$, where $\beta \in (0,1)$ is a reward coefficient, and $C_{i,R(t)}$ is the resource cost of ℓ_i for serving request $R(t)$ concurrently (see (7)). Moreover, the longer the request takes to finish (executing out of SLO λ), the more we should penalize the agent. Similarly, we define the reward of the last MoE layer ℓ_n as

$$r_{n,t} = \begin{cases} -\beta \cdot C_{n,R(t)}, & P95_{r \in R(t)} \tau_r \le \lambda, \\ -\beta \cdot C_{n,R(t)} + \lambda - P95_{r \in R(t)} \tau_r, & \text{o/w.} \end{cases} \tag{13}$$

If the agent fails to reach target SLO λ, the term $\lambda - P95_{r \in R(t)} \tau_r$ penalizes it with negative returns. The objective of the agent is to maximize cumulative reward given by $\sum_{t \in \mathcal{T}} \sum_{\ell_i \in \mathcal{L}} \gamma^{t-1} r_{i,t}$, where $\gamma \in (0,1)$, aligning with our objective (1).

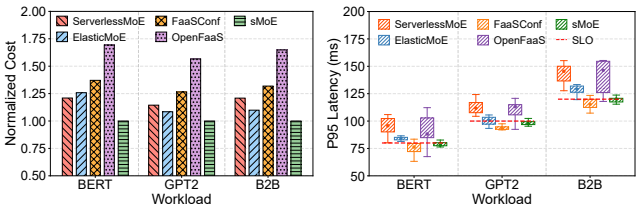
Policy Learning. We employ the Deep Deterministic Policy Gradient (DDPG), a famous off-policy algorithm [25] with actor-critic architecture, for policy learning. At timestep t, the agent independently inputs the state $s_{i,t} = \mathcal{H}_{i,t}$ of each MoE layer ℓ_i and produces a series of actions $a_{i,t} = \{a_{i,j,t} | \forall e_{i,j} \in \mathcal{E}_i\}$ using policy π_θ, where $a_{i,j,t} = \{a_{i,j,t}^{\phi}\}$. The critic $Q_\omega(s_{i,t}, a_{i,t})$ evaluates the action and provides value-based feedback for policy improvement. Temporal-Difference (TD) is used to update the critic parameters ω by sampling tuples (s, a, r) from the replay memory $\mathcal{D}$, and then minimizing the TD loss during training. Here, s, a, and r are the state, action, and reward sets, respectively. Given feedback from the critic, the actor is updated via the deterministic policy gradient:

$$\nabla_\theta J(\pi_\theta) = \mathbb{E}_{s \sim \mathcal{D}} \left[\nabla_a Q_\omega(s,a)|_{a=\pi_\theta(s)} \nabla_\theta \pi_\theta(s) \right]. \tag{14}$$

5 Experimental Evaluation

5.1 Experimental Settings

Implementation. We prototype sMoE on OpenFaaS [16], a widely-used open-source serverless platform. sMoE is deployed on a Kubernetes cluster with 8 nodes, each equipped with 52 CPUs, 128 GB RAM, and an Nvidia RTX 3090 GPU. We disable the default autoscaler in OpenFaaS and use sMoE as its external controller. It periodically reads per-expert and per-layer metrics from Prometheus [26], as well as makes expert-level scaling decisions by calling our GPRL model. Finally, these decisions are applied by patching the corresponding function manifests, thereby taking over resource provisioning from the built-in autoscaler.



(a) Normalized cost. (b) Tail latency.

Figure 7: The performance of sMoE against four baselines.

Workloads. Following prior works [8], we convert three standard Transformer language models into MoE variants.

- **BERT**: a BERT-based encoder model with 12 transformer layers, converted into 12 MoE layers with 4, 6, or 8 experts per layer.
- **GPT2**: a GPT-2-based decoder model with 12 transformer layers, converted into 12 MoE layers with 4 or 6 experts per layer.
- **B2B**: a Bert2Bert model with 12 encoder and 12 decoder layers, converted into 24 MoE layers with 4 experts per layer.

Each expert function can be configured with 6 CPU cores at most and memory limits from 128 MB to 10 240 MB with 64 MB increments. We apply the NVIDIA container toolkit [27] to enable the functions to use the CUDA devices. The GPU share is allocated in units of 0.1 via MPS [28]. Also, we adjust the function replica count from 1 to 8 and concurrency settings ranging from 1 to 10.

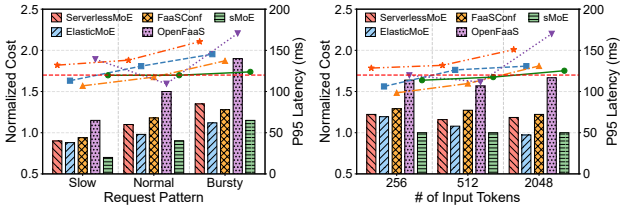
Invocation Traces. To approximate the real-world invocation patterns, we use Azure Function Trace [7], collected from a real production environment over a 14-day period, to generate user requests. We scan the trace and randomly select three different 1-hour traces with typical Coefficient of Variation (CV) of inter-arrival time. Specifically, we define three patterns: slow (CV <1), Normal (1<CV<3), and Bursty (CV>3).

Baselines. We compare sMoE against the following baselines:

- **ServerlessMoE** [8] is a static expert configuration solution for serverless MoE inference, which is reused across requests.
- **ElasticMoE** [29] is an elastic MoE scaler that independently adjusts per-expert configuration based on observed token load.
- **FaaSConf** [30] is a hybrid resource manager for general serverless workflows using multi-agent reinforcement learning.
- **OpenFaaS** [16] is our baseline system with Kubernetes HPA. We randomly select other configurations from the valid space.

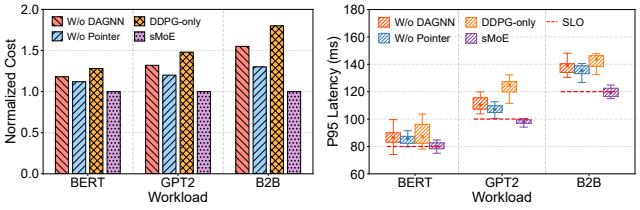
5.2 Evaluation Result

Overall Performance. We first compare sMoE with all baselines under the *bursty* request pattern. Figure 7 reports the normalized total cost and per-window P95 latency, where each workload has its own latency SLO (80 ms for BERT, 100 ms for GPT2, and 120 ms for B2B). As shown, sMoE reduces normalized total cost by 21.4%–39.2% relative to the baselines, while keeping P95 latency within 2.7% of the workload-specific SLO targets. OpenFaaS incurs the highest cost because HPA tends to over-scale replicas under bursty demand. Meanwhile, ServerlessMoE has a moderate cost but frequently violates SLOs because its static expert configuration cannot track popularity shifts. ElasticMoE and FaaSConf reduce SLO violations compared to ServerlessMoE, yet often either exceed the SLO in a noticeable fraction of windows or keep P95 latency far below the



(a) Varying request patterns. (b) Varying Token lengths.

Figure 8: The scalability of sMoE against four baselines.



(a) Normalized cost. (b) Tail latency.

Figure 9: The ablation study of sMoE against three variants.

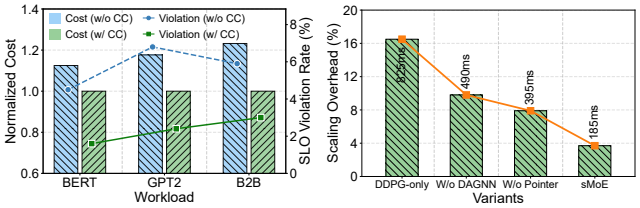


Figure 10: The benefits of scaling concurrency.

Figure 11: The scaling overhead under various variants.

SLO, implying over-provisioning. In contrast, sMoE delivers a more favorable SLO–cost trade-off.

Scalability Evaluation. As shown in Figure 8, sMoE consistently achieves the lowest serving cost while keeping P95 latency close to the SLO across all patterns and input sizes. In contrast, Serverless-MoE becomes increasingly SLO-violating under bursty traffic and longer inputs, indicating that static expert configurations cannot absorb workload variation. ElasticMoE and FaaSConf oscillate between under- and over-provisioning (either exceeding or staying far below the SLO), and OpenFaaS remains the most expensive with highly erratic P95 behavior. That is, sMoE can scale experts more effectively to adapt to both traffic burstiness and input complexity.

Impact of Concurrency Optimization. Figure 10 compares the full system (*w/ CC*) with the variant without concurrency control (*w/o CC*). As shown, *w/ CC* consistently lowers cost and reduces SLO violations by jointly tuning per-expert concurrency with hardware resources. On BERT, *w/o CC* already meets the SLO in most intervals but still incurs noticeable extra cost, whereas on GPT2 and especially B2B it suffers markedly higher violation rates under bursty traffic. This confirms that our joint optimization is critical under dynamic concurrent requests.

Ablation Study. We quantify the contribution of each component in sMoE by comparing the full sMoE with *w/o DAGNN*, *w/o Pointer*, and *DDPG-only*. As shown in Figure 9, sMoE consistently yields the lowest normalized cost and keeps P95 latency closest to each workload's SLO, whereas all variants are more expensive and violate SLOs more often. Removing DAGNN notably increases cost and tail latency, especially on GPT2 and B2B, underscoring the value of topology-aware state representation. Removing the pointer module has a smaller but still visible impact, showing that intra-layer expert coordination helps mitigate stragglers.

Scaling Overhead. As shown in Figure 11, *DDPG-only* incurs the highest overhead and longest decision latency, since it processes a large flat expert state with a monolithic actor. *w/o Pointer* reduces overhead but still requires running DAGNN and per-expert heads, while *w/o DAGNN* is slightly cheaper yet remains above the sMoE. In contrast, sMoE combines DAGNN-based aggregation with a pointer-guided SISO actor to reuse expert embeddings, achieving the lowest control overhead and decision latency and keeping runtime cost within a small fraction of each scaling interval, making it suitable for online serving.

6 Related Work

Serverless Resource Configuration. Serverless computing provides fine-grained, event-driven resources, motivating work on allocation, autoscaling, and cost modeling [6, 12, 31]. For single functions, Bilal et al. [32] examine CPU-memory coupling, and Sizeless [33] predicts near-optimal memory sizes from monitoring data. For workflows, ORION [34] and StepConf [35] optimize DAG-level resource allocation, Aquatope [36] accounts for QoS demands and uncertainty, and FaaSConf [30] co-optimizes scaling and concurrency. However, these techniques operate at function or workflow granularity and do not model MoE-specific scatter-gather behavior or token skew. In contrast, sMoE treats experts and non-MoE segments as a DAG and learns expert-level scaling policies.

MoE Inference and Expert Scaling. MoE inference is explored to handle token routing and expert imbalance. Systems such as Tutel [37], FasterMoE [38], and Lina [39] optimize expert parallelism and communication on GPU clusters. Recent work improves gating and load balancing [40], overlaps communication with computation [20], and explores disaggregated MoE serving [41], but typically assumes a fixed accelerator pool. ElasticMoE [29] adapts per-expert configurations to token load and latency without SLO guarantees. In serverless settings, Liu et al. [8] optimize static expert deployment via Bayesian search over expert layouts. Unlike these approaches, sMoE enables elastic, expert-level scaling under concurrent requests while explicitly targeting end-to-end SLOs.

7 Conclusion

This paper introduces sMoE, a DRL-based expert auto-scaler for serverless MoE inference. sMoE treats the inference pipeline as a DAG, and aggregates cross-layer partial-order dependencies via DAGNN with expert- and layer-level semantics. Then, it uses a layer-wise pointer actor to jointly decide heterogeneous resources and concurrency for each expert. Experimental results show that sMoE can provide expert-level elasticity for serverless MoE inference to minimize inference costs while ensuring SLOs.

Acknowledgments

This research was supported by the Zhejiang Provincial Natural Science Foundation of China under Grant No. LQN26F020008 and Sponsored by CCF-Kuaishou Large Model Explorer Fund (NO. CCF-KuaiShou 2025001).

References

[1] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, et al. Learning transferable visual models from natural language supervision. In *International conference on machine learning*, pages 8748–8763, 2021.

[2] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2020.

[3] Noam Shazeer, Azalia Mirhoseini, Krzysztof Maziarz, Andy Davis, Quoc Le, Geoffrey Hinton, and Jeff Dean. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. *arXiv preprint arXiv:1701.06538*, 2017.

[4] Dmitry Lepikhin, HyoukJoong Lee, et al. Gshard: Scaling giant models with conditional computation and automatic sharding. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2021.

[5] William Fedus, Barret Zoph, and Noam Shazeer. Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity. *Journal of Machine Learning Research*, 23(120):1–39, 2022.

[6] Eric Jonas et al. Cloud programming simplified: A berkeley view on serverless computing. *arXiv preprint arXiv:1902.03383*, 2019.

[7] Mohammad Shahrad et al. Serverless in the wild: Characterizing and optimizing the serverless workload at a large cloud provider. In *USENIX ATC*, pages 205–218, 2020.

[8] Mengfan Liu et al. Optimizing distributed deployment of mixture-of-experts model inference in serverless computing. In *IEEE INFOCOM*, pages 1–10, 2025.

[9] Wentao Liu, Yuhao Hu, Ruiting Zhou, Baochun Li, and Ne Wang. Remoe: Towards efficient and low-cost moe inference in serverless computing. *arXiv preprint arXiv:2512.18674*, 2025.

[10] Yanan Yang, Laiping Zhao, Yiming Li, Huanyu Zhang, Jie Li, Mingyang Zhao, Xingzhen Chen, and Keqiu Li. Infless: a native serverless system for low-latency, high-throughput inference. In *ACM ASPLOS*, pages 768–781, 2022.

[11] Jie Li, Laiping Zhao, Yanan Yang, Kunlin Zhan, and Keqiu Li. Tetris: Memory-efficient serverless inference through tensor sharing. In *USENIX ATC*, 2022.

[12] Cunchi Lv, Xiao Shi, Zhengyu Lei, Jinyue Huang, Wenting Tan, Xiaohui Zheng, and Xiaofang Zhao. Dilu: Enabling gpu resourcing-on-demand for serverless dl serving via introspective elasticity. In *ACM ASPLOS*, pages 311–325, 2025.

[13] Jiacheng Liu, Peng Tang, Wenfeng Wang, Yuhang Ren, Xiaofeng Hou, Pheng Ann Heng, Minyi Guo, and Chao Li. A survey on inference optimization techniques for mixture of experts models. *ACM Computing Surveys*, 58(10):1–37, 2026.

[14] Alexandru Agache, Marc Brooker, Alexandra Iordache, Anthony Liguori, Rolf Neugebauer, et al. Firecracker: Lightweight virtualization for serverless applications. In *USENIX NSDI*, pages 419–434, 2020.

[15] Xiaofei Yue, Song Yang, Liehuang Zhu, Stojan Trajanovski, and Xiaoming Fu. Demeter: Fine-grained function orchestration for geo-distributed serverless analytics. In *IEEE INFOCOM*, pages 2498–2507, 2024.

[16] OpenFaaS Community. OpenFaaS, 2025. https://www.openfaas.com.

[17] Jiaang Duan, Shiyou Qian, Hanwen Hu, et al. PipeCo: Pipelining cold start of deep learning inference services on serverless platforms. *ACM Meas. Anal. Comput. Syst.*, 9(2):1–23, 2025.

[18] Masoud Rahimi Jafari, Jianchang Su, Yifan Zhang, Oliver Wang, and Wei Zhang. PISeL: Pipelining dnn inference for serverless computing. In *ACM CIKM*, pages 1951–1960, 2024.

[19] Veronika Thost and Jie Chen. Directed acyclic graph neural networks. *arXiv preprint arXiv:2101.07965*, 2021.

[20] Yulei Qian, Fengcun Li, Xiangyang Ji, Xiaoyu Zhao, Jianchao Tan, Kefeng Zhang, and Xunliang Cai. EPS-MoE: Expert pipeline scheduler for cost-efficient MoE inference. *arXiv preprint arXiv:2410.12247*, 2024.

[21] Ashish Vaswani, Noam Shazeer, et al. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[22] Oriol Vinyals, Meire Fortunato, and Navdeep Jaitly. Pointer networks. *Advances in neural information processing systems*, 28, 2015.

[23] Qifan Pu, Shivaram Venkataraman, and Ion Stoica. Shuffling, Fast and Slow: Scalable analytics on serverless infrastructure. In *Proc. USENIX Symp. Netw. Syst. Des. Implement.*, pages 193–206, 2019.

[24] Hongzi Mao, Malte Schwarzkopf, Shaileshh Bojja Venkatakrishnan, Zili Meng, and Mohammad Alizadeh. Learning scheduling algorithms for data processing clusters. In *Proc. ACM SIGCOMM Conf.*, pages 270–288, 2019.

[25] Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning, 2019.

[26] Prometheus Community. Prometheus: Monitoring system & time series database, 2025. https://prometheus.io/.

[27] NVIDIA. Nvidia container toolkit, 2025. https://github.com/NVIDIA/nvidia-container-toolkit.

[28] NVIDIA. Nvidia multi-process service (mps), 2025. https://docs.nvidia.com/deploy/mps/index.html.

[29] Gursimran Singh, Timothy Yu, et al. ElasticMoE: An efficient auto scaling method for mixture-of-experts models. *arXiv preprint arXiv:2510.02613*, 2025.

[30] Yilun Wang, Pengfei Chen, Hui Dou, Yiwen Zhang, Guangba Yu, Zilong He, and Haiyu Huang. Faasconf: Qos-aware hybrid resources configuration for serverless workflows. In *Proceedings of the 39th IEEE/ACM International Conference on Automated Software Engineering*, pages 957–969, 2024.

[31] Xiaofei Yue, Song Yang, Fan Li, Liehuang Zhu, Xu Wang, Zhen Feng, and Fernando A Kuipers. Hyfaas: Accelerating serverless workflows by unleashing hybrid resource elasticity. *IEEE Transactions on Parallel and Distributed Systems*, 37(1):272–286, 2025.

[32] Muhammad Bilal, Marco Canini, Rodrigo Fonseca, and Rodrigo Rodrigues. With great freedom comes great opportunity: Rethinking resource allocation for serverless functions. In *Proc. Eur. Conf. Comput. Syst.*, pages 381–397, 2023.

[33] Simon Eismann, Long Bui, Johannes Grohmann, Cristina L. Abad, Nikolas Herbst, and Samuel Kounev. Sizeless: Predicting the optimal size of serverless functions. In *Middleware*, pages 248–259, 2021.

[34] Ashraf Mahgoub et al. ORION and the three rights: Sizing, bundling, and prewarming for serverless DAGs. In *Proc. USENIX Symp. Oper. Syst. Des. Implement.*, pages 303–320, 2022.

[35] Zhaojie Wen, Yishuo Wang, and Fangming Liu. Stepconf: Slo-aware dynamic resource configuration for serverless function workflows. In *IEEE INFOCOM 2022-IEEE conference on computer communications*, pages 1868–1877. IEEE, 2022.

[36] Zhuangzhuang Zhou et al. Aquatope: QoS-and-uncertainty-aware resource management for multi-stage serverless workflows. In *ACM ASPLOS*, pages 1–14, 2022.

[37] Changho Hwang, Wei Cui, Yifan Xiong, Ziyue Yang, Ze Liu, Han Hu, Zilong Wang, Rafael Salas, Jithin Jose, Prabhat Ram, et al. Tutel: Adaptive mixture-of-experts at scale. *Proceedings of Machine Learning and Systems*, 5:269–287, 2023.

[38] Jiaao He, Jidong Zhai, Tiago Antunes, Haojie Wang, Fuwen Luo, Shangfeng Shi, and Qin Li. FasterMoE: Modeling and optimizing training of large-scale dynamic pre-trained models. In *Proceedings of the 27th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPoPP)*, pages 120–134, 2022.

[39] Jiamin Li, Yimin Jiang, et al. Accelerating distributed moe training and inference with lina. In *USENIX ATC*, pages 945–959, 2023.

[40] Haiyang Huang, Newsha Ardalani, Anna Sun, Liu Ke, Hsien-Hsin S Lee, Shruti Bhosale, Carole-Jean Wu, and Benjamin Lee. Toward efficient inference for mixture of experts. *Advances in Neural Information Processing Systems*, 37:84033–84059, 2024.

[41] Ruidong Zhu, Ziheng Jiang, Chao Jin, et al. Megascale-infer: Efficient mixture-of-experts model serving with disaggregated expert parallelism. In *Proceedings of the ACM SIGCOMM Conference*, pages 592–608, 2025.