# Optimizing Distributed Deployment of Mixture-of-Experts Model Inference in Serverless Computing

Mengfan Liu*, Wei Wang†, and Chuan Wu*
*Department of Computer Science, The University of Hong Kong
†Department of Computer Science and Engineering, The Hong Kong University of Science and Technology
Email: ml621@connect.hku.hk, weiwa@cse.ust.hk, cwu@cs.hku.hk

***Abstract*—With the advancement of serverless computing, running machine learning (ML) inference services over a serverless platform has been advocated, given its labor-free scalability and cost effectiveness. Mixture-of-Experts (MoE) models have been a dominant type of model architectures to enable large models nowadays, with parallel expert networks. Serving large MoE models on serverless computing is potentially beneficial, but has been underexplored due to substantial challenges in handling the skewed expert popularity and scatter-gather communication bottleneck in MoE model execution, for cost-efficient serverless MoE deployment and performance guarantee. We study optimized MoE model deployment and distributed inference serving on a serverless platform, that effectively predict expert selection, pipeline communication with model execution, and minimize the overall billed cost of serving MoE models. Especially, we propose a Bayesian optimization framework with multi-dimensional $\epsilon$-greedy search to learn expert selection and optimal MoE deployment achieving optimal billed cost, including: 1) a Bayesian decision-making method for predicting expert popularity; 2) flexibly pipelined scatter-gather communication; and 3) an optimal model deployment algorithm for distributed MoE serving. Extensive experiments on AWS Lambda show that our designs reduce the billed cost of all MoE layers by at least 75.67% compared to CPU clusters while maintaining satisfactory inference throughput. As compared to LambdaML in serverless computing , our design achieves 43.41% lower cost with a throughput decrease of no more than 18.76%.**

## I. Introduction

Serverless computing is a cloud computing paradigm that the cloud providers elastically manage the provisioning of resources (servers, containers, etc.) to deploy services according to their user demand [1]. Serverless computing has been used for serving data analytic applications such as web services [1] [2]. In recent years, there has been an increasing trend in adopting serverless computing for machine learning (ML) services, particularly for model inference. For example, Gillis [3] studies model partitioning and scheduling for large deep neural network (DNN) inference on AWS Lambda [4].

Deploying ML inference services on a serverless platform is more appealing than using traditional GPU/CPU clusters for several reasons. *First*, it frees ML developers from managing hardware resources and virtual machine/container environments, simplifying service deployment and maintenance [5]. *Second*, its pay-as-you-go pricing model ensures cost efficiency by charging only for resources actively used in fine granularity, avoiding unnecessary costs for idle resources [6]. *Third*, serverless functions have been provided to support parallelisms needed for large-scale ML model inference like AWS Lambda Functions [4]. State-of-the-art commercial serverless platforms largely support CPU services [4], [7]–[9]. Though GPU-based model inference has been preferred for high serving performance, using CPUs for inference serving has been a viable alternative, given that high-caliber GPUs are often in shortage, and CPUs are more available and provide substantial cost savings, while being able to meet application service level objectives (SLOs) [10].

The Mixture-of-Experts (MoE) models have been a dominant type of model architectures to enable large models nowadays [11], achieving high model capacity without increasing computation [12] [13]. To build a large model using the MoE architecture, layers in a representative DNN model (e.g., Transformer) are replaced by MoE layers. Each MoE layer includes a gating network and multiple parallel expert networks. During model inference, each input token to an MoE layer is first evaluated by the gating network, which determines the most relevant expert(s) to handle the token [11]. Then the token is routed to the selected expert(s) for computation, and the processing results are aggregated to produce the MoE layer output. MoE models have been widely used to serve various tasks, e.g., SwitchTransformer [11] for text generation.

Serving a large MoE model is resource intensive as it requires substantial memory to deploy the parallel experts. After deployment, devices incur costs even when idle, making GPUs/CPUs more costly than using a serverless platform. We advocate serving MoE models in a CPU-based serverless platform, for labor and cost-efficient management of inference serving, which has been underexplored. The main target for MoE model deployment in a serverless platform is to minimize the billed cost of all MoE layers in serving, for memory usage and execution time of serverless functions that run the MoE layers [4], [7]–[9]. Two major challenges arise for cost-minimal distributed deployment of MoE model inference in a serverless platform.

*First*, serverless functions are typically deployed with memory size configured before the service runs, and the skewed, unknown-beforehand expert selection of input tokens complicates proactive memory configuration of the functions. In MoE serving, some experts (each run as a serverless function) receive many tokens for processing, while others do not. Intuitively, popular experts should be run on serverless functions with larger memory while non-popular ones use

less memory. Existing MoE serving solutions [14] [15] in GPU/CPU clusters decide resource assignment for experts during MoE inference, which is infeasible in serverless platforms. Deploying a serverless function takes several minutes. The first time after a serverless function is deployed, it takes a long time for the function to start execution, due to resource initialization (i.e., the cold start issue) [2]. This would cause long delays in MoE serving if its serverless functions are deployed according to the current demand, degrading the efficiency inference serving. Therefore, the key challenge is to efficiently and accurately estimate expert popularity before the MoE inference service starts in a serverless platform. This enables optimizing memory configurations for serverless functions, thereby decreasing the billed cost of MoE serving.

*Second*, the scatter-gather communication for token-to-expert routing and expert processing result aggregation at MoE layers is time-consuming, that may block subsequent operations as non-MoE layers must wait for all experts to complete their computation and communication [13]. Existing proposals on redesigning MoE scatter-gather communication with pipelining [16] [17] in CPU/GPU clusters are inadequate in serverless platforms. Direct inter-function transfers in a serverless platform are constrained by platform-specified maximum data transfer size (i.e., payload size), while indirect transfers via external storage (e.g., S3 bucket for AWS Lambda [4]) take longer as the data must be saved to external storage and then retrieved. Pipelining data transfer with model execution is infeasible with direct transfers: a serverless function retains no data between invocations (i.e., stateless property) and direct data transfers from other functions require re-invocation of the function each time; model parameters that a serverless function uses are not retained during direct transfers and hence need to be reloaded for each re-invocation, resulting in significant time and memory waste. Indirect transfers between serverless functions rely on external storage, incurring longer communication time and higher cost, adding complexity to pipelining design. This calls for novel communication designs tailored to MoE inference in serverless platforms.

Tackling these challenges, we design a serverless MoE inference solution that effectively predicts expert popularity, pipelines MoE communication with model execution, and optimally deploys MoE model for distributed inference, that minimizes the billed cost of all MoE layers in MoE model serving. We propose a Bayesian optimization (BO) framework with multiple $\epsilon$-greedy search (GS) to learn expert popularity and optimize MoE deployment for billed cost minimization. Our main contributions are summarized as follows:

▷ We design a novel Bayesian decision-making approach for expert selection prediction, including a comprehensive token feature design, a novel posterior calculation approach, and an adjustable key-value dataset table. We analyze the MoE inference process to extract relevant token features. The posterior calculation incorporates real request distributions to refine the posterior using profiled data. The key-value dataset table is adjusted with new key-value pairs according to feedback from model inference, which is used to update the profiled data probabilities and improve prediction accuracy.

▷ We propose several scatter-gather communication designs for a serverless platform: indirect transfers with flexible pipeline operations via external storage, simple indirect transmissions without pipelining, and simple direct invocation of serverless functions without pipelining. As different designs perform the best in terms of execution time under different scenarios, selecting a proper scatter-gather communication design affects billed cost saving in serverless MoE inference.

▷ We formulate optimal deployment of distributed MoE inference on a serverless platform into a mixed-integer quadratically constrained programming (MIQCP) problem, which chooses one of the proposed scatter-gather communication designs, sets memory configurations of expert serverless functions, and determines the number of function replicas. We design an optimal deployment selection (ODS) algorithm, achieving a billed cost of all MoE layers upper bounded by a constant ratio of the optimal solutions.

▷ We propose a BO framework with multiple $\epsilon$-GS to optimize expert selection predictions and distributed deployment of the MoE model. The BO framework iteratively adjusts the key-value dataset table for expert selection prediction using feedback billed cost of all MoE layers in serving. The expert predictions then serve as input for optimal MoE model deployment, which further provides cost feedback for dataset table update. The multiple $\epsilon$-GS balances exploration of different key-value pairs in the dataset table and exploitation of high-performing key-value pairs, in dataset table update.

▷ We implement our designs on AWS Lambda using Pytorch and Optuna packages. Extensive experiments show that our designs reduce the billed cost of all MoE layers by at least 75.67% compared to inference over CPU clusters, while maintaining a satisfactory inference throughput by at least 43.41% compared to LambdaML [18] in serverless computing, with a throughput decrease of at most 18.76%.

## II. Background and Motivation

### A. Serverless Computing

Serverless functions are stateless, that they retain no data from their previous execution. Serverless functions obtain data inputs mainly in two ways: a function can directly transfer output to another function as input when invoking the latter, and a *payload* size limits the maximal data transfer size between functions. When the data exceeds the payload size, external storage is used for relaying data between two functions [1] [3]. External storage can also store other data needed by serverless functions, such as model parameters.

Serverless functions, along with external storage, Docker image manager (e.g., ECR [19] for Amazon Lambda [4]), and serverless function deployment manager (e.g., step function [20] for Amazon Lambda [4]) can be used for ML inference. As shown in Fig. 1, deployment of an MoE model for inference serving on a serverless platform involves several steps. First, the MoE model is partitioned. Next, each model partition is built as a Docker image, which is then pushed to the Docker image manager, with its model parameters stored

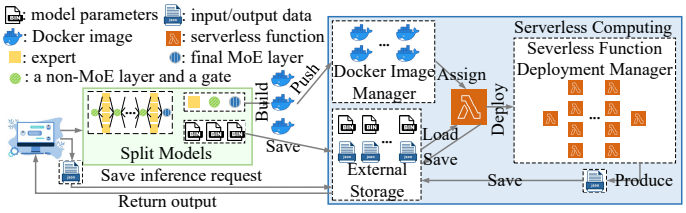



Fig. 1. Overview of MoE model deployment on a serverless platform.

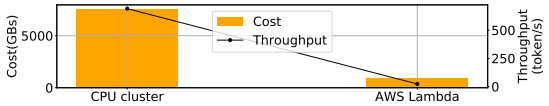



Fig. 2. Billed cost of all MoE layers and inference throughput of a GPT-2-based MoE model.

in external storage. Finally, each Docker image is assigned to a serverless function and these functions are deployed into the serverless platform by a serverless function manager. After deployment, inference requests from service users are stored in external storage and retrieved by the deployed MoE model for inference serving. During inference, each serverless function loads its model parameters and intermediate computation results from external storage, and saves intermediate results back to external storage. Current commercial serverless computing platforms [4], [7]–[9] are CPU-based. Therefore, we focus on CPU-based serverless platforms in this paper.

Commercial serverless computing providers [4], [7]–[9] generally charge users on the used memory during running time of a serverless function at the unit of GB-second (i.e., GBs). We focus on the billed cost of serverless functions as it represents the primary cost for MoE model inference, given a fixed model size and inference request workload.

### *B. MoE Inference*

**Distributed MoE inference.** Data and expert parallelisms in distributed MoE serving necessitate communication across multiple devices for input distribution and output aggregation. FasterMoE [21] applies tensor slicing and pipelining design to overlap all-to-all communication and computation in MoE layers. PipeMoE [16] and MpipeMoE [17] study the pipeline degree for tensor slicing to adaptively pipeline communication and computation. The designs rely on the hardware architecture of GPU/CPU clusters such as the memory layout , and thus cannot be adopted on serverless platforms.

**ML services in serverless platforms.** Extensive research focuses on optimizing ML model serving in serverless platforms. Amps-inf [22] studies model partitioning to minimize inference costs. Ali et al. [23] design a request batch queue to reduce costs. Gillis [3] adopts model partitioning and scheduling to accelerate DNN model inference, while ServerlessLLM [24] focuses on fast multi-tier checkpoint loading and optimized startup-time scheduling. DNN serving on serverless platforms has been proven effective in meeting serving SLOs and handling varying workloads. However, there is a lack of designs for efficient MoE inference on a serverless platform.

### *C. Opportunities and challenges*

**Opportunity: MoE model inference in a serverless platform can bring substantial cost reduction and satisfactory inference performance**, compared to CPU cluster-based inference serving. On a serverless platform, experts can be individually assigned to serverless functions with different memory size configurations, so that more resources are rented for workload-heavy experts and less for workload-light experts. Serverless functions are only billed on assigned memory when executing. However, the cost of a CPU cluster typically depends on the amount of resources over a fixed coarse-grained period (e.g., per month or per hour), so that costs may still be incurred for idle resources. Fig. 2 shows the billed cost of all MoE layers and inference throughput, when a GPT-2-based MoE model serves 10,240 tokens from the Enwiki8 [25] dataset. The CPU cluster uses two 64-core AMD EPYC CPUs with 512GB of DRAM for the entire MoE model. Each serverless function for MoE inference on AWS Lambda is allocated 3008 MB of memory. The billed cost on AWS Lambda is significantly lower than on the CPU cluster. The throughput of MoE inference on the serverless platform is 22.9 tokens per second, substantially exceeding the human reading speed of 3.3 tokens per second [26]. Hence, serverless-based MoE inference presents a viable solution. Two challenges exist on enabling efficient deployment and serving of an MoE model on a serverless platform.

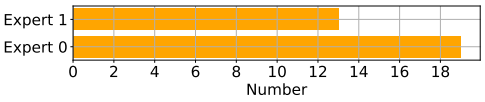
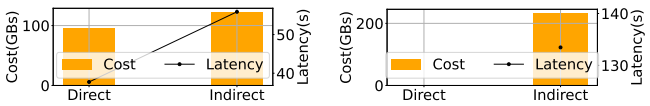



Fig. 3. Number of tokens with token ID 10424 (from the Enwiki8 dataset) routed to different experts at the 2nd MoE layer in Bert-based MoE model.

**Challenge 1: Skewed, unknown-beforehand expert popularity prevents proactive, accurate memory configuration of serverless functions.** The gating network routes tokens to experts based on token features, e.g., position in the inference request sequence, meaning of a word token, its roles in the sequence (subject, object, verb, etc.) [27]. Expert selection is unknown before actual token processing by the gating network. Dynamic resource allocation, which decides computational resources for each expert based on real-time expert popularity during inference, has been advocated for MoE serving in GPU/CPU clusters [14] [21]. Serverless functions require a long time to deploy (e.g., 1 minute or longer) and to start execution (e.g., 5 seconds or longer) after deployment [2]; on-demand dynamic function redeployment is largely infeasible, which would substantially slow down inference serving. This necessitates proper configuration of memory sizes when deploying serverless functions before service starts, requiring prior knowledge of expert popularity. Improper memory configuration may either fail to meet the memory demand during inference, or incur higher billed costs due to unnecessary memory usage. FlexMoE [28] and Prophet [29] use average historical expert popularity to adjust resources among experts, but achieve low prediction accuracy as the token-to-expert relationship is not explored. Cong et al. [30] propose an LSTM-based algorithm trained on historical token-to-expert mapping for expert selection prediction, but are memory-intensive and require a long training time. Lina [27]




(a) 256 tokens (b) 2560 tokens

Fig. 4. Billed cost of all MoE layers and end-to-end inference time of a Bert-based MoE model on AWS Lambda (tokens from Enwiki8 dataset; payload size 6MB).

predicts expert popularity using the maximum a posteriori probability based on historical token-to-expert mapping and token ID information. Token ID is insufficient to fully identify a token in token-to-expert mappings, as tokens with the same token ID may be routed to different experts at an MoE layer, as illustrated in Fig. 3.

**Challenge 2: MoE scatter-gather communication renders performance bottlenecks in serverless MoE inference.** In each MoE layer, scatter-gather communication often renders a bottleneck as the non-MoE layer must wait for all experts to complete their computation and communication. The direct and indirect inter-function communication on a serverless platform results in different billed costs and inference time. In Fig. 4, MoE scatter-gather communication via indirect transfers incurs higher cost and longer inference time than direct transfers when serving a 256-token batch, due to the additional function running time required for storing data in external storage and retrieving data as input. Direct transfers cannot be adopted when serving a 2560-token batch as the payload size is exceeded; the serving cost under indirect transfers is very high. While pipelining communication with computation can typically alleviates this bottleneck in GPU/CPU clusters, pipelining in a serverless platform needs to take its direct or indirect data transfer modes into consideration, and be carefully designed to improve efficiency in a serverless platform.

## III. Design

### A. System overview

We consider distributing an MoE model in a serveless platform for inference. We adopt expert parallelism [13], [27] by assigning each expert to a serverless function. Model parallelism is adopted for non-MoE parts of the model with each non-MoE layer assigned to a serverless function. We mainly focus on the MoE layers since the non-MoE layers are traditional DNNs extensively studied [3], [31].

We propose a Bayesian Optimization (BO) framework to learn expert selection and optimal deployment of the MoE model for inference serving. The goal is to minimize overall billed cost of all MoE layers in serving. The BO framework consists of a *Feedback Processor* to adjust expert selection prediction, an *Expert Selection Predictor*, and a *Policy Maker*. An illustration is given in Fig. 5. The expert selection predictor provides expert predictions of inference tokens from a real-world dataset on an inference task. This prediction is based on the posterior distribution calculated from these tokens and profiled data. The profiled data records the number of times each token-to-expert mapping occurs across at least 100

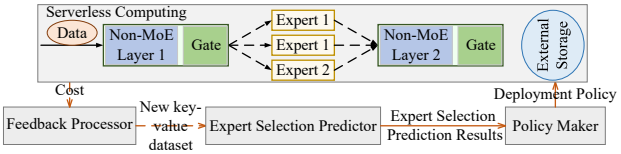



Fig. 5. System Overview.

samples from the same real-world dataset, organized in a key-value table. With the predictions, the policy maker decides how to deploy the MoE model, configures the memory size of each expert serverless function and adopts our scatter-gather communication design. The feedback processor adjusts the key-value pairs in the profiled table for improving expert selection prediction, using feedback of the billed cost of all MoE layers in serving. In each BO iteration, expert predictor is adjusted, the policy maker decides optimal deployment of the MoE model, and the billed cost is collected for feedback processor's key-value table adjustment.

When the BO algorithm converges, the MoE model is deployed according to the learned expert popularity and optimal deployment policy, and serves real-world inference requests.

### B. Expert selection prediction

We carefully design token features by investigating more token information during MoE inference. As Transformer models are typically the backbone of MoE models [11], we focus on Transformer-based MoE models.

For Transformer-based MoE models, token processing mainly occurs in the embedding, encoder and decoder layers. In the embedding layers and feed-forward networks in the encoder and decoder layers, each token is embedded with its own information and its position (e.g., word embedding and position embedding). Thus, the token ID and position ID can be extracted as token features. Each multi-head attention layer in the encoder and decoder layers contains multiple self-attentions. Each self-attention calculates the Query, Key, and Value of the tokens to capture dependencies between tokens in the token sequence. The dependencies are quantified by the softmax attention scores, which indicate the relevance of each token to the others. For each token, we extract these dependencies as a token feature. The dependencies for each token are simplified as the token ID of the token with the highest sum of softmax attention scores across all self-attentions at a multi-head attention layer, referred to as the attention ID. The attention ID may vary before different MoE layers, aligning with the diverse expert popularities at different MoE layers. Therefore, the token features include the token ID, the position ID, and the attention ID.

Our expert prediction through the BO framework is learned on profiled data, which records the token-to-expert mappings on at least 100 samples from a real-word dataset of inference task. The profiled data are organized in a key-value table where the keys are token-to-expert mappings and the values denote their occurrence counts. Especially, we design a new posterior calculation method and use the maximum posterior approach to predict expert selection for new tokens, where the posterior represents the probability of an expert given a token [27].

Assume $\mathbf{f}$ is the token feature vector of a token, in which $\mathbf{f}_1$ is the token ID, $\mathbf{f}_2$ is the position ID and $\mathbf{f}_3$ is the attention ID. The posterior given the token is $\mathcal{P}(\mathbb{N}_{e,i}|\mathbf{f})$ with $e \in \mathbb{E}$, where $\mathbb{N}_{e,i}$ is the $i$-th expert in the expert set $\mathbb{N}_e$ at MoE layer $e$ and $\mathbb{E}$ is the set of MoE layers in the MoE model. For a new token that has not undergone MoE inference, its feature $\mathbf{f}'_1$ is known but $\mathbf{f}'_3$ is unknown. The probability of $\mathbf{f}_2$ at any position is uniform, and the probability of $\mathbf{f}_3$ at any value can be approximated by the probability of $\mathbf{f}_1$ at that value, as the attention ID $\mathbf{f}_3$ is the token ID with the highest attention scores. We can obtain all probabilities related to $\mathbf{f}'_1$ from the profiled data, the uniform probability $\mathcal{P}'(\mathbf{f}_2)$ of $\mathbf{f}_2$ at any value and the probability $\mathcal{P}'(\mathbf{f}_3)$ of $\mathbf{f}_3$ at any value from tokens in the same real-world dataset, that have not undergone MoE inference.

To leverage all token features for identifying a token effectively, we use Bayes' theorem to design a new posterior calculation method. The Bayes' theorem $\mathcal{P}(\mathbb{N}_{e,i}|\mathbf{f}) = \mathcal{P}(\mathbf{f}|\mathbb{N}_{e,i})\mathcal{P}(\mathbb{N}_{e,i})/\mathcal{P}(\mathbf{f})$ describes how the posterior of an expert selection given a token $\mathcal{P}(\mathbb{N}_{e,i}|\mathbf{f})$ is updated with the prior of the expert $\mathcal{P}(\mathbb{N}_{e,i})$, the prior of the token $\mathcal{P}(\mathbf{f})$, and the likelihood of the token given the expert $\mathcal{P}(\mathbf{f}|\mathbb{N}_{e,i})$. We can involve $\mathcal{P}'(\mathbf{f}_2)$ and $\mathcal{P}'(\mathbf{f}_3)$ into the likelihood $\mathcal{P}(\mathbf{f}'_1|\mathbb{N}_{e,i})$, through the joint probability $\mathcal{P}(\mathbf{f}'_1, \mathbb{N}_{e,i})$. For simplicity, we multiply $\mathcal{P}'(\mathbf{f}_2)$ and $\mathcal{P}'(\mathbf{f}_3)$ with $\mathcal{P}(\mathbf{f}'_1, \mathbb{N}_{e,i})$ as involvement. Hence, the designed posterior calculation method is given by:

$$\mathcal{P}(\mathbb{N}_{e,i}|\mathbf{f}'_1) = \int_{\mathbf{f}_2}\int_{\mathbf{f}_3} \mathcal{P}^*(\mathbb{N}_{e,i}|\mathbf{f}'_1, \mathbf{f}_2, \mathbf{f}_3)\frac{\mathcal{P}^*(\mathbf{f}'_1, \mathbf{f}_2, \mathbf{f}_3)\mathcal{P}'(\mathbf{f}_3)}{\mathcal{P}^*(\mathbf{f}'_1, \mathbf{f}_2)}d\mathbf{f}_3 \frac{\mathcal{P}^*(\mathbf{f}'_1, \mathbf{f}_2)\mathcal{P}'(\mathbf{f}_2)}{\mathcal{P}^*(\mathbf{f}'_1)}d\mathbf{f}_2, \forall e \in \mathbb{E}, \forall i \in \mathbb{N}_e, \tag{1}$$

where $\mathcal{P}^*(\cdot)$ represents the probability calculated from profiled data in the key-value dataset.

Therefore, we use the maximum a posterior method to predict the expert selection for a token with $\mathbf{f}'_1$:

$$\hat{i}_e = \text{argmax}_{i \in \mathbb{N}_e} \mathcal{P}(\mathbb{N}_{e,i}|\mathbf{f}'_1), \forall e \in \mathbb{E}, \tag{2}$$

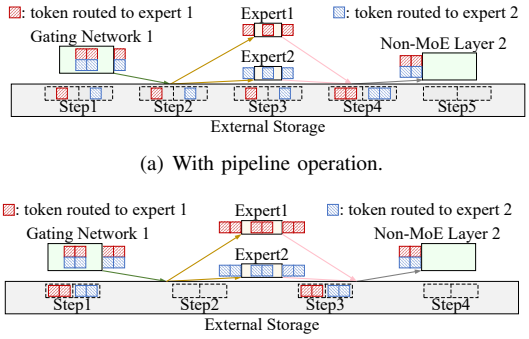
where $\hat{i}_e$ is the predicted expert at MoE layer $e$. Eq. (2) can be readily extended to top-$k$ expert selection prediction.

### C. Scatter-gather communication design

We design pipelined scatter-gather communication at the MoE layer in a serverless platform for better cost reduction.

For a batch of tokens to serve, we set a pipeline degree $\beta$ to split the batch for each expert into minibatches, where $\beta$ is the maximal minibatch size. At each MoE layer, the gating network routes the splitted minibatches to each expert and the next non-MoE layer gathers processed minibatches from each expert. If the minibatch size exceeds the payload limit, indirect transfer is used; otherwise, direct transfer is adopted.

Pipelining is only achievable with indirect transfer via external storage on a serverless platform as serverless functions are stateless. We use one block time to represent the maximal overlap time of the indirect upload of a minibatch and the download and calculation of the next minibatch. The block time is determined by the pipeline degree $\beta$: a larger $\beta$ results in fewer minibatches and longer block time. The benefits of pipelining can be reduced by the access delay to external storage. We design three scatter-gather communication methods tailored for MoE inference on a serverless platform. Let $a_e \in \mathbb{A} = \{1, 2, 3\}$ denote three communication methods at MoE layer $e$. We allow all experts at an MoE layer to use the same method to simplify implementation. Three options exist.

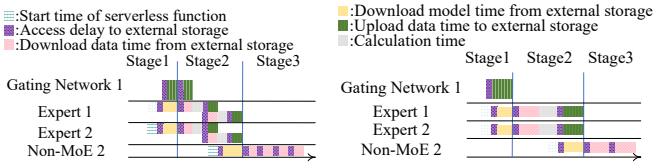



(a) With pipeline operation.

(b) Without pipeline operation.

Fig. 6. Scatter-gather communication with indirect transfers through external storage: (a) with pipelining; (b) without pipelining. Pipeline degree $\beta$ is 2.


Fig. 7. Scatter-gather communication with direct function invocation.




(a) With pipeline operation.

(b) Without pipeline operation.

Fig. 8. Scatter-gather communication time with indirect transfers through external storage: (a) with pipelining; (b) without pipelining.

- In the first option ($a_e = 1$), the gating network splits each expert's input into minibatches and sends them to external storage; at each expert, downloading a minibatch from external storage and calculating this minibatch are overlapped with uploading the previous processed minibatch to external storage. As shown in Fig. 6(a), after splitting input data into minibatches of one token each, two minibatches for two experts are stored in external storage (step 1); these two minibatches are downloaded from external storage and calculated by two expert serverless functions, while the next two minibatches or processed minibatches are stored in external storage (steps 2-3); all processed minibatches are stored in external storage (step 4) and then the next non-MoE layer downloads all minibatches from external storage (step 5).
- In the second and the third options, data is indirectly ($a_e = 2$) or directly ($a_e = 3$) transferred without pipeline operations between the gating network and parallel experts, as well as the parallel experts and the next non-MoE layer, as shown in Fig. 6(b) and Fig. 7.

The total MoE layer time varies with different communication designs. Fig. 8 and Fig. 9 illustrate the execution time. We will carefully make the choices in our distributed MoE deployment problem.

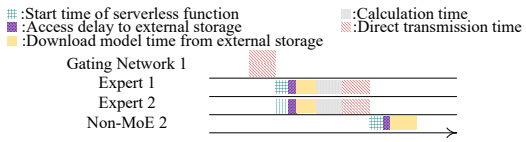



Fig. 9. Scatter-gather communication time with direct function invocation.

### D. MoE model deployment

The policy maker decides optimal MoE model deployment by making the following decisions:

• Memory size configuration for each serverless function. Let $\mathbb{M}$ be the set of memory size options for each serverless function (e.g., from 128MB to 3008MB on AWS Lambda) $x_{e,i,j} \in \{0,1\}$ denotes if the $j$-th option in set $\mathbb{M}$ is selected for expert $i$ in MoE layer $e$ (1) or not (0). The processing time of one token at expert $i$ in MoE layer $e$, is given by:

$$t^{\text{cal}}_{e,i} = s_{e,i} \sum_{j=1}^{|\mathbb{M}|} x_{e,i,j} U_j, \forall e \in \mathbb{E}, \forall i \in \mathbb{N}_e, \tag{3}$$

where $s_{e,i} \in \{0,1\}$ denotes if the expert is selected (1) or not (0) for the token (given based on expert selection prediction), and $U_j$ is the time to process one token in an expert using the $j$-th memory size option with the serverless function.

• Expert replication. As the maximal memory size of each serverless function is limited, it is possible that it still takes a long time for a popular expert to process tokens routed to it. Given that an end-to-end inference time target should be met in inference serving, we further consider replicating serverless functions of experts and allow expert replicas to run in parallel for toke processing. Let $y_{e,i,g} \in \{0,1\}$ denote if expert $i$ in MoE layer $e$ has $g$ serverless function replicas (1) or not (0), where $g = 1, ..., G$ with $G$ as the maximal possible replica number. The number of tokens routed to one replica is $r_{e,i} = \sum_{g=1}^{G} y_{e,i,g} d_{e,i}/g$, where $d_{e,i}$ denotes the number of tokens routed to all replicas of the expert. Let $D^{\text{in}}$ be the size of one token and $D^{\text{p}}$ be the maximal payload size in the serverless platform. When $r_{e,i} D^{\text{in}} > D^{\text{p}}$, direct transfer is not feasible at MoE layer $e$ (i.e, $a_e$ should be 1 or 2).

• Scatter-gather communication method and parameter ($\beta$). We seek to minimize the total billed cost of all MoE layers. The billed cost of the gating network can be ignored here, as it affects the cost of all MoE layers little: the memory size of serverless functions for gating networks does not depend on expert popularity, and the impact of communication methods on the gating network's execution time is also reflected in expert execution time. Hence, we focus on experts for the total billed cost of all MoE layers. The billed cost of MoE layer $e$ is then given by $c_e = (a_e-2)(a_e-3)c_{1,e} + (a_e-1)(a_e-3)c_{2,e} + (a_e-1)(a_e-2)c_{3,e}$, where the billed cost of MoE layer $e$ under the communication method $a_e$ is:

$$c_{a_e,e} = \sum_{i \in \mathbb{N}_e} s_{e,i} t_{a_e,e,i} \sum_{j=1}^{M} x_{e,i,j} \mathbb{M}_j, \forall e \in \mathbb{E}, \forall a_e \in \mathbb{A}, \tag{4}$$

where $\mathbb{M}_j$ represents the $j$-th memory size option in the set $\mathbb{M}$, and $t_{a_e,e,i}$ is the execution time of all replicas of the expert:

$$t_{a_e,e,i} = \sum_{g=1}^{G} y_{e,i,g} g t^{\text{rep}}_{a_e,e,i}, \forall e \in \mathbb{E}, \forall i \in \mathbb{N}_e, \forall a_e \in \mathbb{A}, \tag{5}$$

where $t^{\text{rep}}_{a_e,e,i}$ is the execution time of one replica of the expert, related to the selected scatter-gather communication method. We give formulas of three cases in obtaining $t^{\text{rep}}_{a_e,e,i}$.

For pipelined indirect transfer, $t^{\text{rep}}_{1,e,i} = T^{\text{h,E}}_{e,i} + t^{\text{nblk}}_{e,i} + \beta t^{\text{blk}}_{e,i}, \forall e \in \mathbb{E}, \forall i \in \mathbb{N}_e$, where $T^{\text{h,E}}_{e,i} = \frac{P_{e,i}}{B^{\text{s}}} + T^{\text{dl}} + T^{\text{str}}$ consists of the warm start time $T^{\text{str}}$, the access delay to external storage $T^{\text{dl}}$ and the model download time $\frac{P_{e,i}}{B^{\text{s}}} + T^{\text{dl}}$ with bandwidth $B^{\text{s}}$ between the external storage and the function as well as the parameter size $P_{e,i}$ of the expert. $t^{\text{nblk}}_{e,i} = T^{\text{dl}} + \lceil \frac{r_{e,i}}{\beta} \rceil (\frac{D^{\text{out}}}{B^{\text{s}}})$ includes the time to upload the last minibatch to external storage and $D^{\text{out}}$ is the size of the processed result of one token by an expert. $t^{\text{blk}}_{1,e,i} = T^{\text{dl}} + \beta \max\{\frac{D^{\text{in}}}{B^{\text{s}}} + t^{\text{cal}}_{e,i}, \frac{D^{\text{out}}}{B^{\text{s}}}\}$ denotes one worst-case block time. The other two cases (i.e., $t^{\text{rep}}_{2,e,i}$ and $t^{\text{rep}}_{3,e,i}$) are derived based on Fig. 8(b) and Fig. 9. Details can be found in our technical report [32] and omitted due to space limit.

The latency, from the earlist time point when expert serverless functions start or the gating network starts to transfer each expert's input, to the latest time when the next non-MoE layer finishes downloading the processed results of all experts from external storage or model parameters with direct transfer, is referred to as MoE-E2E latency $t^{\text{lat}}_{1,e}$. We have $t^{\text{lat}}_{1,e} = \max\{t^{\text{S12}}_{1,e}, T^{\text{load}}_e\} + t^{\text{S3}}_{1,e}, \forall e \in \mathbb{E}$, where $T^{\text{load}}_e$ includes the time to start the serverless function of the non-MoE layer and download the model parameters, and $t^{\text{S12}}_{1,e}$ and $t^{\text{S3}}$ are derived based on Fig. 8(a) with superscript $S$ as the Stage. The other two cases (i.e., $t^{\text{lat}}_{2,e}$ and $t^{\text{lat}}_{3,e}$) are derived based on Fig. 8(b) and Fig. 9. Details can be found in our technical report [32] and omitted due to space limit.

The MoE-E2E latency at MoE layer $e$ is given by $t^{\text{lat}}_e = (a_e-2)(a_e-3)t^{\text{lat}}_{1,e} + (a_e-1)(a_e-3)t^{\text{lat}}_{2,e} + (a_e-1)(a_e-2)t^{\text{lat}}_{3,e}$.

**Optimal MoE deployment problem:** We formulate optimal deployment of MoE model inference in a serverless platform to minimize the billed cost of all MoE layers (i.e., $\sum_{e \in \mathbb{E}} c_e$), by jointly deciding the communication method (i.e., $a_e$), memory size configurations (i.e. $x_{e,i}$), expert replication (i.e. $y_{e,i}$), and parameter $\beta$ for pipelined scatter-gather communication.

$$\min \sum_{e \in \mathbb{E}} c_e \tag{6a}$$

$$\text{subject to} \quad 3-5, \tag{6b}$$

$$P_{e,i} + M^{\text{itrm}}_{e,i} + r_{e,i}(D^{\text{in}} + D^{\text{out}}) \leq \sum_{j=1}^{|\mathbb{M}|} x_{e,i,j} \mathbb{M}_j, \forall e \in \mathbb{E}, \forall i \in \mathbb{N}_e \tag{6c}$$

$$T^{\text{head}} + T^{\text{tail}} + \sum_{e \in \mathbb{E}} (t^{\text{lat}}_e + T^{\text{NE}}_e) \leq T^{\text{limit}}, \tag{6d}$$

$$1 \leq \lceil \frac{r_{e,i}}{\beta} \rceil \leq \max_{i \in \mathbb{N}_e, e \in \mathbb{E}} r_{e,i}, \tag{6e}$$

$$(a_e - 3)(r_{e,i} D^{\text{in}} - D^{\text{p}}) \leq 0, \forall e \in \mathbb{E}, \forall i \in \mathbb{N}_e, \forall a_e \in \mathbb{A}, \tag{6f}$$

$$\sum_{j=1}^{|\mathbb{M}|} x_{e,i,j} = 1, \sum_{g=1}^{G} y_{e,i,g} = 1, \forall e \in \mathbb{E}, \forall i \in \mathbb{N}_e, \tag{6g}$$

$$x_{e,i,j} \in \{0,1\}, \forall e \in \mathbb{E}, \forall i \in \mathbb{N}_e, \forall j \in \mathbb{M}, \tag{6h}$$

$$y_{e,i,g} \in \{0,1\}, \forall e \in \mathbb{E}, \forall i \in \mathbb{N}_e, \forall g \in \{1, \ldots, G\}, \tag{6i}$$

$$\beta \in \mathbb{Z}, \tag{6j}$$

Here $M_{e,i}^{\text{itrm}}$ is the memory size of intermediate results during an expert's inference. $T^{\text{limit}}$ is the time limit of end-to-end MoE model inference (i.e., serving SLO). $T^{\text{head}}$ is the execution time of the first non-MoE layer. $T^{\text{tail}}$ is the execution time of the last non-MoE layer, excluding the data transmission time. $T_e^{\text{NE}}$ is the processing time of non-MoE layer $e$ with the subsequent gating network. (6c) specifies the memory limit of each serverless function, (6d) gives the end-to-end inference time target of the MoE model, (6e) limits the maximal number of tokens in each block in calculating the worst-case total time, and (6f) prohibits direct tranfers when the payload size is below transferred data size between the gating network and experts, and between experts and the next non-MoE layer.

## IV. The BO Framework

We now present our BO algorithm using the BO framework to learn expert selection predictions and optimize MoE model deployment, together with an efficient ODS algorithm to derive optimal MoE model deployment by the policy maker.

### A. Optimal MoE Deployment Algorithm

Given expert selection results, the optimal MoE deployment problem in (6) is NP-hard [33] due to non-linearity and discretized variables. We linearize max functions in (6a), (6d) and (6e) by adding auxilliary variables (e.g., $\max_{h\in\mathbb{H}}\{h\}$ can be linearized as $\phi \geq h$, $\forall h \in \mathbb{H}$). Then we solve each MIQCP by a solver [34], respectively, and obtain costs $c_{1,e}$, $c_{2,e}$ and $c_{3,e}$, $\forall e \in \mathbb{E}$, from the three solutions. Based on these solutions, we design an Optimal Deployment Selection (ODS) algorithm to decide $a_e$ for each MoE layer, as follows.

For each MoE layer $e$, we select the communication method $\hat{a}_e$ with the lowest cost, and calculate the MoE-E2E latency $\hat{t}^{\text{lat}}_{\hat{a}_e,e}$. If the new MoE-E2E latency satisfies the end-to-end inference time constraint, the optimal deployment policy for the MoE model is obtained; otherwise, we identify the MoE layer $\tilde{e}$ with the highest latency, set the cost of the corresponding scatter-gather communication $\tilde{a}_{\tilde{e}}$ to infinity, and then iteratively decide $\hat{a}_e$ for each MoE layer. At most $2|\mathbb{E}|$ iterations are needed, as $3|\mathbb{E}|$ solutions of $\hat{a}_e$, $\forall e \in \mathbb{E}$, are provided, and selecting $|\mathbb{E}|$ solutions excludes up to $2|\mathbb{E}|$ other solutions. If all costs $c_{a_e,e}$ become infinity, it implies that mixing different communication methods across different MoE layers do not work. In this case, we return the optimal deployment policy with the lowest cost with all MoE layers using the same scatter-gather communication method.

### B. BO algorithm

BO is a statistical approach for global optimization of a black-box function, including an objective, variables, a surrogate function to simulate the objective, and an acquisition method to update the variables. Our BO algorithm adjusts the key-value table, recomputes expert selection predictions and distributes MoE model deployment. The objective of our BO is to minimize the billed cost of all MoE layers in the serverless platform. The variables are $Q$ key-value pairs in the key-value table $\Omega(\cdot)$. The surrogate function uses a Gaussian process to simulate the cost of all MoE layers deployed by the policy maker based on expert selection prediction. For the acquisition method, we design a decaying multi-dimensional $\epsilon$-greedy search (GS) to set the variables for the next BO iteration. Traditional single-dimension $\epsilon$-GS, as an acquisition function, decays with each iteration and balances exploration and exploitation of variables by selecting the best variable with probability $1-\epsilon$ and exploring new variable with probability $\epsilon$ [35]. However, as multiple key-value pairs in key-value table are adjusted together in a single BO iteration, these pairs can be viewed as multi-dimensional variables in our BO, which makes a single-dimension $\epsilon$ insufficient to balance exploration and exploitation in all dimensions. We extend $\epsilon$ to a multi-dimensional vector $\epsilon \in \mathbb{R}^Q$.

In BO learning, we obtain the objective on several batches of inference data from an open-source real-world dataset [25], [36], [37] (different datasets can be used for different MoE inference tasks). When inaccurate expert selection prediction occurs compared to the ground truth in the profiled data, we set a limited range of key-value pairs to update as $\mathbb{L}$, where $\mathbb{L}$ includes all positive integers for values and all token-to-expert mappings with token IDs limited to those present in these batches for keys. We set a ratio $\mu$, and slow down the decay of the split $\epsilon_{1:\mu Q}$ of vector $\epsilon$ from the first dimension to the $\mu Q$-th dimension by multiplying $\epsilon_{1:\mu Q}$ with a factor greater than 1. The 1st to $\mu Q$-th key-value pairs are then updated using $\epsilon_{1:\mu Q}$ by adjusting the values of keys in $\mathbb{L}$, allowing for more exploration on current low-performing key-value pairs. Meanwhile, the $\mu Q+1$-th through the last key-value pairs in the variables are updated by either adjusting the values of keys in $\mathbb{P}$ or by creating new key-value pairs in $\mathbb{P}$ using $\epsilon_{\mu Q+1:Q}$, which enriches the key-value table. Here $\mathbb{P}$ is similar to $\mathbb{L}$ but extensively includes all token IDs assigned by the tokenizer.

The BO algorithm with multi-dimensional $\epsilon$-GS is given as follows. In the $\tau$-th BO iteration, $\epsilon$ decays by being divided by $1+\rho\tau$ with $\rho > 0$. The dataset table $\Omega_\tau$ is updated with key-value pairs $\{\mathbf{z}_{\tau-1,q}, v_{\tau-1,q}\}_{\forall q\in\{1,\ldots,Q\}}$, where $\mathbf{z} = \{\mathbf{f}, e, i\}$. Then expert selection is predicted. The policy maker produces the optimal deployment policy using the ODS algorithm. For the $j$-th batch in BO learning, at expert $i$ in MoE layer $e$, if the difference between predicted count $r^{e,i}$ and real count $R_{e,i}^{\text{real}}$ of tokens assigned to one replica of this expert exceeds a constant $\alpha > 0$, token IDs $\mathbf{f}'_{j,0}$ of the $j$-th batch are recorded for $\mathbb{L}$ to adjust and three cases are discussed: (i) if the minimal memory $M^{\text{real}}$ for real expert popularity exceeds the memory configuration of serverless functions, $\rho_1 < \rho$ is used to decrease the decay rate $\epsilon_{\tau,1:\mu Q}$, and $n_{e,i}^{\text{new}}$ is calculated to replicate expert $i$ for $n_{e,i}^{\text{new}}$ times to satisfy the minimal memory $M^{\text{real}}$; (ii) if the size of transferred tokens exceeds the payload size under direct transfer, $\rho_2 < \rho_1$ is used to decrease the decay rate $\epsilon_{\tau,1:\mu Q}$, and $n_{e,i}^{\text{new}}$ is calculated to replicate expert $i$ for $n_{e,i}^{\text{new}}$ times to ensure that the data size transferred to each replica does not exceed the payload size $D^{\text{p}}$ ; (iii) if all constraints in (6) are satisfied, $\rho_3 < \rho_2$ is used to decrease the decay rate $\epsilon_{\tau,1:\mu Q}$, and we do not replicate expert $i$ to

avoid increasing cost . The billed cost of all MoE layers $c_{\tau,j}$ is computed on the $j$-th batch data on the derived MoE model deployment $MoE_\tau$. Next, the historical set $\mathbb{B}$ in BO to record variables and objectives is updated, and the key-value pairs to adjust is updated by $\epsilon$-GS over $\mathbb{B}$ and the range of variables $\mathbb{P}$ and $\mathbb{L}$. BO iterations repeat until the change of the minimal billed cost of all MoE layers within $\lambda$ consecutive iterations is below the threshold $\zeta$.

### C. Theoretical Analysis

**Theorem 1.** *ODS Alg. produces feasible MoE deployment in $O(|\mathbb{E}|)$ time, which achieves a billed cost of all MoE layers upper bounded by a constant ratio of the cost of optimal solutions of (6).*

**Theorem 2.** *BO Alg. converges when the BO iteration index satisfies $\tau > \frac{1+\rho}{\rho-\rho_1}(1 - \frac{\delta}{\max_{q\in\{1,\ldots,Q\}} \epsilon_{0,q}})$ with an abitrary small positive constant $\delta < \max_{q\in\{1,\ldots,Q\}}\epsilon_{0,q}$.*

Theorem 1 indicates that the time complexity of ODS Alg. scales linearly with the number of MoE layers. Theorem 2 shows that the BO algorithm converges to a constant bound. The proof can be found in our technical report [32] and omitted here due to space limit.

## V. Evaluation

### A. Experimental Setup

**Testbed.** We run the experiments on AWS Lambda [4]. To build MoE layer images, we use a Dockerfile to define the environment with Python 3.8 and include packages such as *torch* and *transformers*. We implement the BO algorithm with package *optuna* [38] and MIQCP solvers with package *gurobi* [34]. We use two S3 buckets of size 512MB each for external storage. We adopt 14 discrete memory size configurations for each serverless function: [128, 768, 960, 1152, 1344, 1536, 1728, 1920, 2112, 2304, 2496, 2688, 2880, 3008] MB. We set the maximal possible number of expert replicas as 8.

**MoE Models.** Three common transformer-based dense language models are converted to MoE models with all MLP layers after attention layers converted to MoE layers and a gating network of a linear layer: (1) Bert [39]: a 12-layer encoder model with 110 million parameters, converted to 12 MoE layers, with each MoE layer having 4, 8, or 16 experts; (2) GPT2 [40]: a 12-layer decoder model with 1.5 billion parameters, converted to 12 MoE layers, with each MoE layer having 4 experts; (3) Bert2Bert [41]: a 12-layer encoder-decoder model with 247 million parameters, converted to 24 MoE layers, with each MoE layer having 4 experts. We run the fill-mask task [42] on Enwik8 [25] and CCnews [43] datasets, and the translation task [44] on the Wmt19 [37] dataset for the BERT model. We conduct the text generation task on the Enwik8 and Lambda [36] datasets for the GPT-2 model and on the Enwik8 dataset for Bert2Bert model.

### B. Expert selection prediction

We first evaluate accuracy of expert selection prediction learned by our BO framework, by calculating the average absolute difference per expert between the real and predicted

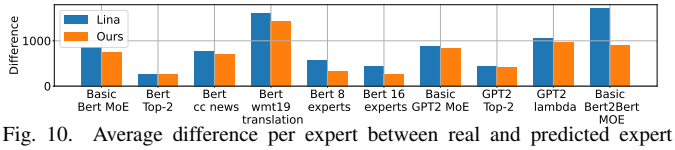
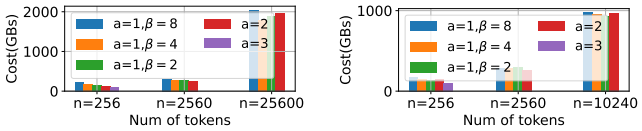
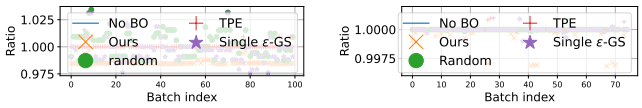



Fig. 10. Average difference per expert between real and predicted expert selection numbers under different MoE models, datasets and tasks.



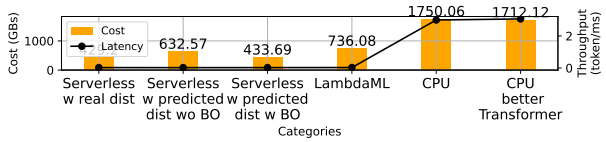

(a) Bert cost. (b) GPT2 cost.

Fig. 11. Billed cost of MoE layers with different scatter-gather communication methods on AWS Lambda.

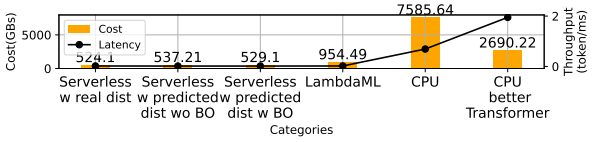



(a) Bert cost. (b) GPT2 cost.

Fig. 12. Ratio of billed cost optimized by BO with different acquisition functions over no BO.




(a) Bert.




(b) GPT2.

Fig. 13. Billed cost of all MoE layers and inverse of throughput under different expert selection distributions on AWS Lambda and CPU clusters.

counts of tokens assigned to each expert. Fig. 10 shows the difference across different MoE models, datasets, and tasks. For a task on a dataset, we use 95% of this training set for profiling and evaluate the difference on 10,240 tokens in the testing set. Basic Bert MoE represents the Bert MoE model with 4 experts per MoE layer and top-1 routing for the fill-mask task on Enwiki8. Basic GPT2 MoE and Basic Bert2Bert MoE cases are similar. Other cases are variations based on these basic setups, e.g., GPT2 Lambda denotes the GPT2 basic setup but changes the dataset to Lambda [36]. Across all cases, our method outperforms expert prediction in Lina [27], as Lina only uses token ID as token feature while our method incorporates token ID, position ID, and attention ID to capture additional information for more accurate profiled probabilities. Compared to top-1 routing, the results of top-2 routing show that increasing the value of $k$ in top-$k$ gating significantly improves the prediction accuracy, as routing to more experts allows prediction mistakes in one expert to be corrected by the other. When the number of experts increases, the average prediction difference decreases.

### C. Scatter-gather communication

We next evaluate performance of our different scatter-gather communication designs. We allocate 3008MB of memory to each serverless function and use no expert replicas. Fig. 11 shows the billed cost of MoE layers under different communication methods. The results verify that the optimal scatter-gather communication method varies depending on the number of tokens. For 256 tokens, the direct transfer performs best for both Bert MoE and GPT2 MoE. As the number of tokens increases, either pipelined or non-pipelined indirect transfers may perform better, while direct transfer becomes impractical due to payload size limitations.

### D. ODS algorithm

We deploy MoE model for inference using 10,240 tokens on AWS Lambda. We check the billed cost of all MoE layers using our ODS algorithm, an MIQCP method and a random selection method, under different inference throughput targets. The MIQCP method uses one MIQCP solver to directly solve (6), and the random method randomly selects the communication method at each MoE layer. We set the target throughput by dividing 10240 tokens by the end-to-end latency limit specified in (6). The time limit for searching the optimal solution using the MIQCP approach is set to 180s; for the ODS algorithm with three MIQCP solvers, the search time limit is set to 60s. Our ODS algorithm outperforms other methods. At higher target throughputs, the MIQCP method fails to derive an optimal solution within 180s.

### E. BO algorithm

For BO learning, we use 10,240 tokens from the Enwiki8 dataset to simulate the inference requests, and set $Q = 1000$ key-value pairs to update in each BO iteration. After BO learning, we test on 100 batches of inference requests from the testing dataset with batch size of 10240 tokens. Due to the long deployment time on AWS Lambda, we use simulation for this set of evaluations. Fig. 12 shows the ratio of the billed cost optimized by BO with different acquisition functions, to that of no BO, respectively. No BO means that we do not use the BO algorithm to adjust our expert predictor, and the expert selection is predicted by the unadjusted predictor. The random method randomly adjusts key-value pairs, the single $\epsilon$-greedy sampler uses the same $\epsilon$ for all dimensions of the variables, and the TPE [45] method samples on promising regions of variable range based on probabilistic modeling. Our multi-dimension $\epsilon$-GS performs best in terms of the billed cost for both Bert MoE and GPT2 MoE models.

### F. Algorithm overhead

We dissect the execution time of the expert selection predictor, the ODS algorithm and the BO algorithm. For expert selection predictor, the time of profiling 100 batches of data is around 28.89 seconds, and the prediction time on 10 batches is around 20.31 seconds. The execution time of the ODS algorithm with three MIQCP solvers is around 2.27s. Our BO algorithm requires around 62.15s per iteration and converges in around 1257.89s.

### G. Overall performance

We then deploy MoE models on AWS Lambda and a CPU cluster for inference serving of 10,240 tokens. The CPU cluster consists of two 64-core AMD EPYC CPUs with 512GB of DRAM. Fig. 13 shows comparisons under different expert selection distributions (regarding the count of tokens assigned to each expert) among: (1) *Serverless with predicted distribution optimized by BO:* the optimal MoE deployement produced by our BO framework; (2) *Serverless with real expert selection distribution:* the optimal MoE deployment produced based on ground truth of expert selections in the MoE inference; (3) *Serverless with predicted distribution without BO*: the optimal MoE deployment produced using predicted expert selection that is not adjusted by the BO algorithm; (4) *LamdaML [18]*, which uses the maximum memory allocation for each serverless function on AWS Lambda (3008MB) for inference serving, requires no expert prediction, and uses no replicas for each expert; (5) *CPU:* the MoE model is deployed in the CPU cluster, with all experts at each MoE layer executing concurrently, requiring no expert prediction; (6) *CPU better-Transformer:* MoE deployment in the CPU cluster accelerated by the CPU inference optimization betterTransformer [46], through sparsity and fused kernels as Flash Attention.

For both Bert MoE and GPT2 MoE, our serverless MoE inference design consistently results in lower billed costs, as compared to MoE inference on the CPU cluster. Specifically, serverless MoE inference with predicted expert selection reduces the billed cost by at least 75.67% compared to CPU cluster-based serving. The throughput in serverless-based MoE serving remains significantly above human reading levels of 3.3 tokens per second [26]. The lower throughput in serverless MoE serving compared to CPU cluster-based serving is mainly because non-MoE layer computation is limited to the 3008MB memory size of each serverless function, which is far less than the 512GB available in a common CPU cluster. Among serverless options, the predicted expert selection distribution optimized by BO outperforms both non-BO methods and over-provisioning with LambdaML. The BO-optimized expert distribution not only reduces the billed cost by at least 43.41% compared to LambdaML with at most an 18.76% decrease in throughput, but also closely aligns with the cost of deployment using the real expert distribution.

## VI. Conclusion

We study optimized MoE model deployment on a serverless platform to minimize the overall billed cost of serving MoE models. We propose a BO framework with multi-dimensional $\epsilon$-GS to learn expert selections. We design a novel approach for expert selection prediction, propose scatter-gather communication designs, and design an ODS algorithm.


## Acknowledgment

This work was supported in part by grants from Hong Kong RGC under the contracts C7004-22G (CRF), C6015-23G (CRF), 17204423 (GRF), 16210822 (GRF) and T43-513/23-N (TRS).

## REFERENCES


[1] C. Jin, Z. Zhang, X. Xiang, S. Zou, G. Huang, X. Liu, and X. Jin, "Ditto: Efficient serverless analytics with elastic parallelism," in *Proceedings of the ACM SIGCOMM 2023 Conference*, 2023, pp. 406–419.

[2] H. Zhang, Y. Tang, A. Khandelwal, J. Chen, and I. Stoica, "Caerus:{NIMBLE} task scheduling for serverless analytics," in *18th USENIX Symposium on Networked Systems Design and Implementation (NSDI 21)*, 2021, pp. 653–669.

[3] M. Yu, Z. Jiang, H. C. Ng, W. Wang, R. Chen, and B. Li, "Gillis: Serving large neural networks in serverless functions with automatic model partitioning," in *2021 IEEE 41st International Conference on Distributed Computing Systems (ICDCS)*. IEEE, 2021, pp. 138–148.

[4] Aws lambda. [Online]. Available: https://aws.amazon.com/lambda/

[5] A. Mampage, S. Karunasekera, and R. Buyya, "A holistic view on resource management in serverless computing environments: Taxonomy and future directions," *ACM Computing Surveys (CSUR)*, vol. 54, no. 11s, pp. 1–36, 2022.

[6] S. Kounev, N. Herbst, C. L. Abad, A. Iosup, I. Foster, P. Shenoy, O. Rana, and A. A. Chien, "Serverless computing: What it is, and what it is not?" *Communications of the ACM*, vol. 66, no. 9, pp. 80–92, 2023.

[7] Google cloud functions. [Online]. Available: https://cloud.google.com/functions/

[8] Microsoft azure. [Online]. Available: https://learn.microsoft.com/azure/

[9] Alibaba cloud. [Online]. Available: https://www.alibabacloud.com/

[10] P. He, S. Zhou, C. Li, W. Huang, W. Yu, D. Wang, C. Meng, and S. Gui, "Distributed inference performance optimization for llms on cpus," *arXiv preprint arXiv:2407.00029*, 2024.

[11] 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.

[12] N. Du, Y. Huang, A. M. Dai, S. Tong, D. Lepikhin, Y. Xu, M. Krikun, Y. Zhou, A. W. Yu, O. Firat *et al.*, "Glam: Efficient scaling of language models with mixture-of-experts," in *International Conference on Machine Learning*. PMLR, 2022, pp. 5547–5569.

[13] S. Rajbhandari, C. Li, Z. Yao, M. Zhang, R. Y. Aminabadi, A. A. Awan, J. Rasley, and Y. He, "Deepspeed-moe: Advancing mixture-of-experts inference and training to power next-generation ai scale," in *International Conference on Machine Learning*. PMLR, 2022, pp. 18 332–18 346.

[14] J. He, J. Qiu, A. Zeng, Z. Yang, J. Zhai, and J. Tang, "Fastmoe: A fast mixture-of-expert training system," *arXiv preprint arXiv:2103.13262*, 2021.

[15] M. Zhai, J. He, Z. Ma, Z. Zong, R. Zhang, and J. Zhai, "{SmartMoE}: Efficiently training {Sparsely-Activated} models through combining offline and online parallelization," in *2023 USENIX Annual Technical Conference (USENIX ATC 23)*, 2023, pp. 961–975.

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

[17] Z. Zhang, D. Yang, Y. Xia, L. Ding, D. Tao, X. Zhou, and D. Cheng, "Mpipemoe: Memory efficient moe for pre-trained models with adaptive pipeline parallelism," in *2023 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. IEEE, 2023, pp. 167–177.

[18] Lambdaml. [Online]. Available: https://github.com/DS3Lab/LambdaML

[19] Ecr. [Online]. Available: https://aws.amazon.com/ecr/

[20] Step function. [Online]. Available: https://aws.amazon.com/step-functions/

[21] J. He, J. Zhai, T. Antunes, H. Wang, F. Luo, S. Shi, and Q. 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*, 2022, pp. 120–134.

[22] J. Jarachanthan, L. Chen, F. Xu, and B. Li, "Amps-inf: Automatic model partitioning for serverless inference with cost efficiency," in *Proceedings of the 50th International Conference on Parallel Processing*, 2021, pp. 1–12.

[23] A. Ali, R. Pinciroli, F. Yan, and E. Smirni, "Batch: machine learning inference serving on serverless platforms with adaptive batching," in *SC20: International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, 2020, pp. 1–15.

[24] Y. Fu, L. Xue, Y. Huang, A.-O. Brabete, D. Ustiugov, Y. Patel, and L. Mai, "{ServerlessLLM}:{Low-Latency} serverless inference for large language models," in *18th USENIX Symposium on Operating Systems Design and Implementation (OSDI 24)*, 2024, pp. 135–153.

[25] Enwik8. [Online]. Available: http://prize.hutter1.net/

[26] K. Kotani, T. Yoshimi, and H. Isahara, "A machine learning approach to measurement of text readability for efl learners using various linguistic features." *Online Submission*, 2011.

[27] J. Li, Y. Jiang, Y. Zhu, C. Wang, and H. Xu, "Accelerating distributed {MoE} training and inference with lina," in *2023 USENIX Annual Technical Conference (USENIX ATC 23)*, 2023, pp. 945–959.

[28] X. Nie, X. Miao, Z. Wang, Z. Yang, J. Xue, L. Ma, G. Cao, and B. Cui, "Flexmoe: Scaling large-scale sparse pre-trained model training via dynamic device placement," *Proceedings of the ACM on Management of Data*, vol. 1, no. 1, pp. 1–19, 2023.

[29] W. Wang, Z. Lai, S. Li, W. Liu, K. Ge, Y. Liu, A. Shen, and D. Li, "Prophet: Fine-grained load balancing for parallel training of large-scale moe models," in *2023 IEEE International Conference on Cluster Computing (CLUSTER)*. IEEE, 2023, pp. 82–94.

[30] P. Cong, A. Yuan, S. Chen, Y. Tian, B. Ye, and T. Yang, "Prediction is all moe needs: Expert load distribution goes from fluctuating to stabilizing," *arXiv preprint arXiv:2404.16914*, 2024.

[31] F. Romero, Q. Li, N. J. Yadwadkar, and C. Kozyrakis, "{INFaaS}: Automated model-less inference serving," in *2021 USENIX Annual Technical Conference (USENIX ATC 21)*, 2021, pp. 397–411.

[32] Our technique report. [Online]. Available: https://arxiv.org/abs/2501.05313

[33] S. Elloumi and A. Lambert, "Global solution of non-convex quadratically constrained quadratic programs," *Optimization methods and software*, vol. 34, no. 1, pp. 98–114, 2019.

[34] Gurobi optimization. [Online]. Available: https://www.gurobi.com/

[35] G. De Ath, R. M. Everson, J. E. Fieldsend, and A. A. Rahat, "$\epsilon$-shotgun: $\epsilon$-greedy batch bayesian optimisation," *arXiv preprint arXiv:2002.01873*, 2020.

[36] Lambda. [Online]. Available: https://huggingface.co/datasets/cimec/lambada

[37] Wmt19. [Online]. Available: https://huggingface.co/facebook/wmt19-en-de

[38] Optuna. [Online]. Available: https://optuna.org/

[39] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," *arXiv preprint arXiv:1810.04805*, 2018.

[40] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, I. Sutskever *et al.*, "Language models are unsupervised multitask learners," *OpenAI blog*, vol. 1, no. 8, p. 9, 2019.

[41] C. Chen, Y. Yin, L. Shang, X. Jiang, Y. Qin, F. Wang, Z. Wang, X. Chen, Z. Liu, and Q. Liu, "bert2bert: Towards reusable pretrained language models," *arXiv preprint arXiv:2110.07143*, 2021.

[42] Fill-mask task. [Online]. Available: https://huggingface.co/tasks/fill-mask

[43] Ccnews. [Online]. Available: https://huggingface.co/datasets/cc_news

[44] Translation task. [Online]. Available: https://huggingface.co/docs/transformers/tasks/translation

[45] J. Bergstra, R. Bardenet, Y. Bengio, and B. Kégl, "Algorithms for hyper-parameter optimization," *Advances in neural information processing systems*, vol. 24, 2011.

[46] better transformer. [Online]. Available: https://pytorch.org/blog/a-better-transformer-for-fast-transformer-encoder-inference/