Efficient Deep Ensemble Inference via Query Difficulty-dependent Task Scheduling

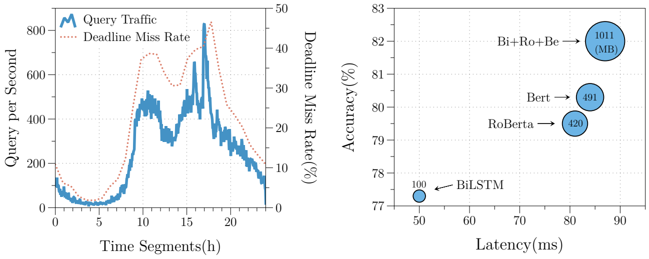
Zichong Li, Lan Zhang, Mu Yuan, Miaohui Song, Qi Song
University of Science and Technology of China
Hefei, China
lzc123@mail.ustc.edu.cn, zhanglan@ustc.edu.cn, ym0813@mail.ustc.edu.cn, songmiaohui@mail.ustc.edu.cn, qisong09@ustc.edu.cn

***Abstract*—Deep ensemble learning has been widely adopted to boost accuracy through combing outputs from multiple deep models prepared for the same task. However, the extra computation and memory cost it entails could impose an unacceptably high deadline miss rate in latency-sensitive tasks. Conventional approaches, including ensemble selection, focus on accuracy while ignoring deadline constraints, and thus cannot cope with input space with high dimension and bursty query traffic. This paper explores redundancy in deep ensemble model inference and presents *Schemble*, a query difficulty-dependent task scheduling framework. *Schemble* treats ensemble inference progress as multiple base model inference tasks and schedules tasks for queries based on their difficulty and queuing status. We evaluate *Schemble* on real-world datasets, considering intelligent Q&A system , video analysis and image retrieval as the running applications. Experimental results show that *Schemble* achieves a 5x lower deadline miss rate and improves the accuracy by 32.9% given deadline constraints.**

***Index Terms*—Ensemble learning, Deep learning, Efficient inference, Task scheduling**

I. Introduction

Deep learning techniques have been adopted in various data systems for data analysis and better service quality [1–3]. Certain aspects, such as model architecture and training data, are essential for deep learning models to be accurate. In practice, however, the performance of an individual model is often limited by low-quality training data and inadequate architecture and hyperparameters. Ensemble learning [4] is proposed to combine decisions from multiple models, named base models, prepared for the same task, so as to improve the overall performance. Many efforts [5, 6] have been devoted to ensembling traditional machine learning models, such as XGBoost [5] and Randomforest [6]. There are also studies investigating the theoretical properties of ensemble learning. For example, Hansen and Salamon [7] proves that in the binary classification task, if the votes from the base models are independent of each other and have accuracies higher than 0.5, overall accuracy could approximate one as the ensemble size grows. Recently, there is a growing trend to ensemble deep learning models [2, 8–10]. They have gained significant improvement in a variety of tasks, including speech recognition [10], video analysis [2], disease diagnose [3], etc..



(a) Query traffic and deadline miss rate at different time segment.

(b) Performance of an ensemble model and base models.

Fig. 1: Performance of an example deep ensemble model (BiLSTM+RoBerta+Bert). (a) One-day query traffic and deadline miss rate of the deep ensemble model in a real intelligent Q&A system. Deadlines are set to 100ms after query arrivals. (b) The performance comparison between the ensemble model and three base models.

Although deep ensemble techniques improve accuracy, such improvements come at a cost in terms of additional computational and memory overhead. We investigate the effects of deep ensemble inference on latency-sensitive tasks, and identify a problem that deep ensemble inference could easily lead to a high deadline miss rate, the ratio of samples that miss given deadlines. Fig. 1a presents a one-day query load recorded from a real intelligent Q&A system and the average deadline miss rate of a deep ensemble model at different time slots. The deadline miss rate is strongly correlated to the query load and reaches 45% during the traffic burst. In online query systems, deadline misses would reduce customer satisfaction and even be fatal. For example, the auto-driving system [11] should respond to stochastic query traffic within very short delays, where deadline misses could lead to accidents and be life-threatening.

Deadline misses are mainly due to the following reasons. Typically, to run an ensemble model, a system loads all its base models and executes them in parallel to reduce latency. Taking models for text matching as an example, Fig. 1b shows that the ensemble improves the accuracy but also incurs longer latency, which is slightly larger than the slowest base model. Hence, more complex structures and synchronous execution of base models lead to a higher deadline miss rate. Moreover, the required memory increases linearly with the ensemble size. A single query will occupy all base models, i.e., a lot of memory, and let subsequent queries wait in a queue. Since many systems only provide limited resources [11, 12], when a large number of queries arrive quickly, traditional execution of large deep ensemble models can cause severe queue blocking, resulting in a very high deadline miss rate.

In this work, we focus on deep ensemble inference for latency-sensitive tasks and propose a new execution mechanism to meet the deadlines of all queries while maximizing the inference accuracy. We first explore the redundancy in deep ensemble inference by evaluating the performance of every combination of the base models in Fig. 1b on datasets from a real Q&A system. We discover that 78.3% of samples can be predicted correctly by any of the three base models if we take the ensemble's outputs as the ground truth. Only less than 11% of samples require executing all three models to get correct predictions, which indicates a large number of redundant executions in deep ensemble inference. Existing practices eliminate the redundancy by shrinking ensemble size [13–15] or dynamically selecting models based on input features [16, 17]. However, current approaches assume base models are run in serial and seek to control the cumulative runtime on offline tasks instead of response latency for online tasks. Thus, they fail to reduce the deadline miss rate effectively. Besides, most studies focus on traditional ML tasks whose input space's dimension is much lower.

To eliminate redundant executions, we break down an ensemble inference into multiple base model inference tasks and dynamically schedule these tasks for each arriving query to meet its deadline. Scheduling includes choosing different combinations of base models for arriving queries and deciding the execution order of inference tasks. Intuitively, the harder the sample, the more base models are needed for ensembling. For example, on machine translation, the complete ensemble model can accurately translate long and complex sentences, but it is unnecessary to execute all base models for easy sentences such as 'Thank you'. By scheduling fewer and faster models for easy samples, we could increase the system throughput and reduce the deadline miss rate with minor accuracy loss. As an example, if there are three base models and two relatively easy queries, normally the second query has to wait for the first one to finish. If we schedule the first model for the first query and the other two models for the second, we can process them simultaneously and get two results with significantly shorter delays. Based on the idea, we present *Schemble*, a query difficulty-dependent task scheduling framework for efficient deep ensemble inference, which accurately predicts query difficulty and schedules base model inference tasks accordingly.

Schemble addresses two core technical challenges. *First*, the input space's dimension in deep learning is usually large; thus it is very challenging to estimate the difficulty of input samples efficiently. To this end, we introduce *discrepancy score* to measure inputs' difficulty and train a lightweight network to predict discrepancy scores for newly arrived queries. *Second*, base model inference tasks need to be selected and scheduled in an online manner to maximize accuracy under deadline constraints. The huge search space of all possible selections of model sets and execution orders for all queries makes it nontrivial to determine the optimal scheduling plan. To cope with this, we break the online scheduling problem into local subproblems and design a near-optimal dynamic programming-based algorithm. We conduct a theoretical analysis of the competitive ratio of the proposed algorithm. In summary, the main contributions are as follows.

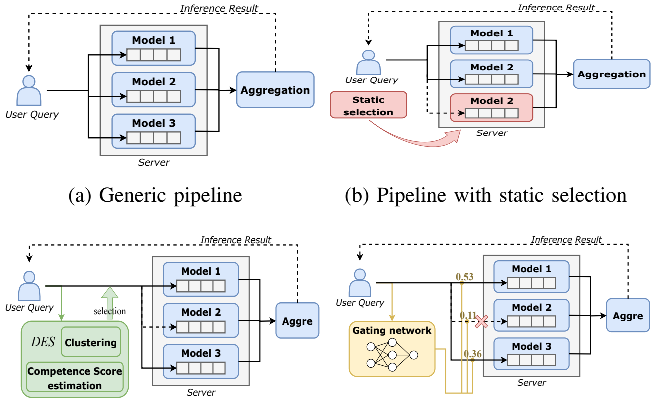
- We identify a novel problem, i.e., efficient deep ensemble inference for latency-sensitive tasks, which maximizes the inference accuracy under deadline constraints for online queries through dynamically scheduling base model inference tasks. We explore inputs' difficulty and analyze the scheduling problem's complexity and optimality.
- We present *Schemble*, a novel query difficulty-dependent task scheduling framework for efficient deep ensemble inference. We propose a lightweight network to predict discrepancy scores that measures inputs' difficulty and design a dynamic programming-based algorithm to achieve efficient inference with limited resources.
- We evaluate *Schemble* on various applications, including intelligent Q&A, video analysis and image retrieval, using both simulated and real-world query traces. Evaluation results show that *Schemble* achieves a 5x lower deadline miss rate and improves the accuracy by 32.9% given deadline constraints in text matching task.

II. Related Work

We categorize related work as follows.

a) Static ensemble selection: Static method selects the same subset of models for inference on every query [14, 18, 19] to reduce model execution, without considering differences between the inputs. The memory cost of the ensemble could be saved by only deploying the chosen subset of models. Caruana et al. [19] proposed a greedy algorithm that selects the model that could improve the overall accuracy most in every iteration until the resource limit is reached. Li et al. [14] theoretically analyzed relations between base model diversity and overall performance, and designed a diversity-regularized selection algorithm. Visentini et al. [18] introduced F-score to rank base models and select models with the highest scores. Static ensemble selection speeds up inference by reducing the ensemble size and could improve performance when bad models exist in the model pool. However, the ensemble size of deep ensembles(around 3) is much smaller than their traditional ensemble problem (50-200). In deep ensemble scenarios, static selection could be accomplished by greedy search at a low cost. Moreover, removing models from deep ensembles results in a bad trade-off between accuracy and computation overhead. It either brings huge accuracy losses with low deadline miss rates or minor ones with high miss rates.

b) Dynamic ensemble selection(DES): DES method selects different models subsets dynamically based on the inputs features[16, 17, 20], similar to our idea. They stated that models specialize in different regions in the input space, which is called models' preferences. They proposed methods to learn models' preferences to select models that is more credible (more likely to output a correct prediction) for each input. The dynamic enables DES to be more flexible than static ensemble selection. DES reduces the total computation overhead on a



(a) Generic pipeline

(b) Pipeline with static selection

(c) Pipeline with DES

(d) Pipeline with Gating

Fig. 2: Deep ensemble inference pipelines with existing approaches

per-query level. However, existing works on DES did not address the high deadline miss rate problem for two main reasons. Firstly, the objectives of these works are not the deadline miss rate but the accuracy or the offline cumulative runtime. We must take arrival times and queuing status into account with a scheduling algorithm to lower the miss rate effectively. Secondly, the base models they considered are mostly traditional ML models and shallow networks, whose preferences are simple and could be nicely captured by simple ML algorithm. Deep learning model in our problem possesses a much more complicated preference space that varies significantly with random seeds. DES methods couldn't capture deep models' preferences and thus result in a vast accuracy loss.

Besides static and dynamic ensemble selection, some works on aggregation mechanisms such as Gating [21, 22] could also be employed to select models for execution. Gating is a popular technique used in MoE [23] that utilizes neural network gates to aggregate base models' outputs. An extra gating network is trained for each base model to output weights for averaging models' outputs. By learning to assign high weights to credible base models, Gating achieves better performance than uniform averaging. We could select a subset of models for execution by thresholding the aggregation weights. As a straightforward solution that also applies neural networks for selection, we discuss the difference between Gating and our method in SectionV-C and compare them thoroughly in the experiments.

Other studies designed machine learning serving systems with optimization on ensemble inference [24, 25]. For example, Crankshaw et al. [25] introduced a module to monitor changes in data distribution and modify the ensemble model adaptively. Lee et al. [24] provided an excellent platform for deploying and maintaining multiple models. These works focus on stability and accuracy rather than per-instance latency. Therefore, we refer to them as orthogonal works.

III. Preliminary

We start with basic terminology and the inference pipelines of a deep ensemble model.

A. Deep ensemble inference

A typical deep ensemble consists of multiple deep learning models and an aggregation module. The deep learning models, also called the base models in the ensemble, perform inference on samples and outputs prediction. The aggregation module combines all predictions to gain better results. Fig. 2a displays an inference pipeline of a deep ensemble with three base models. Before serving, we first employ all base models on the server waiting for inference tasks. Each model executes one task at a time and maintains its own queue to store pending tasks. Then the user could send queries to the system asking for inference results. In a general ensemble inference pipeline, one query would assign an inference task to each base model, and models execute the tasks in parallel. The aggregation module would collect their predictions after all models finish the inference. Under the resource limit that we can not deploy more base models, we would get a high deadline miss rate when lots of queries arrive in a short period of time due to queue blocking. Note that the execution time of the aggregation module is neglectable compared to base models' inference.

B. Ensemble selection

As discussed in SectionII, existing works aim to reduce computational overheads of the ensemble by statically and dynamically selecting subsets of models for execution. We further describe and illustrate how their approaches modulate the pipeline with figures.

Static ensemble selection selects the same subset of models for all queries. Thus, we could spare memory by deleting unchosen models and utilize it with replicas of the chosen models. Fig. 2b display the inference pipeline with static selection that chooses model 1 and model 2. A replica of model 2 is deployed to harness the remaining memory. In practice, thanks to the small ensemble size of the deep ensemble, we are able to find an optimal deployment plan for static selection by greedy search.

Dynamic ensemble selection(DES) selects subsets of models based on input features. Hence, we still need to deploy all base models on the server. DES learns models' preferences, measured by competence score, to determine which subset of models could give accurate predictions on the current query. In general, DES algorithms consist of three steps. First, in the training phase, a clustering method is applied to divide the input space into several regions. Then the algorithm estimates a competence score for each model in each region, representing how credible the model is. Existing works utilize models' accuracy profiling in the region [16], probabilistic models [26], or even neural networks to calculate the score [17]. In the inference phase, the arrived query will trigger a selection mechanism to choose a subset of models based on the estimated competence scores in the region where the query belongs, as shown in fig. 2c. The query would then assign inference tasks to the subset of models DES selects.

The gating method [21] is similar to DES, except that it directly employs neural networks to learn models' preferences, which is called the gating network. The gating network takes the query as the input and outputs a scalar for every base model. In the training phase, the gating network's parameter is optimized by approximating the weighted average of all base models' predictions to the ground truth labels. The gating network would learn to assign large weights to credible models for each query. In the inference phase, we select subsets of models by filtering models with low gating weights, as shown in fig. 2d.

While the static method is inflexible and results in huge accuracy loss, other existing approaches also select subsets of models dynamically, similar to our idea. However, they ignore two main problems. First, preferences of deep learning models are hard to capture. As we will show in section V-C, preferences of deep learning models are complicated and have a huge variance. Learning it with a lightweight neural network is no better than approximating a high variance noise. Also, clustering used in DES does not adapt to high-dimensional inputs. To cope with it, instead of learning models' preferences, we proposed to distinguish between inputs with different difficulties. Second, they ignore the scheduling problem. They both selects model only based on models' preferences, regardless of the queue status. For example, assume there is a pending query and model 1 has the highest credibility on it. Existing approaches assign tasks to model 1 even when its queue is already stuffed. Our framework would select models taking queue status into account with a task scheduling algorithm, and hence achieves a better trade-off between computational overhead and accuracy.

IV. Method Overview

Consider a typical deep ensemble inference system with multiple base models, as shown in Fig. 3. The original pipeline executes all base models for each arrived query, resulting in severe deadline misses. We aim to reduce the deadline miss rate through dynamically scheduling[1] inference tasks for queries. The proposed framework, *Schemble*, executes base models based on query difficulty and queuing status. *Schemble* maintains an extra query buffer and adds three additional components to the original pipeline. The query buffer stores queries that arrive when all base models are busy, which enable *Schemble* to schedule inference tasks with knowledge of subsequent queries. Existing approaches only select model subsets based on current query. The Discrepancy Score Prediction module and the Task Scheduler module process queries in the buffer and offer an execution plan. Once there are idling models, the scheduler will send query inference tasks to the base models according to the plan. The Missing Value Filling module would fill up the missing outputs of unexecuted models for final aggregation. The functions of the added modules are summarized as follows:

- *Discrepancy Score prediction*: This module applies an efficient neural network to predict queries' difficulty measured by the discrepancy score. Paired with offline accuracy profiling, it estimates the accuracy of different combinations of base models on the current query.
- *Task Scheduler*: This module schedules inference tasks for queries based on the estimated accuracy of model combinations and queuing status. It maximizes inference accuracy and ensures queries are completed by the deadlines.
- *Missing Value Filling*: Since fewer inference tasks are assigned, we cannot get outputs from every base model for some queries. This module handles the missing values, ensuring that the aggregation model could work properly.

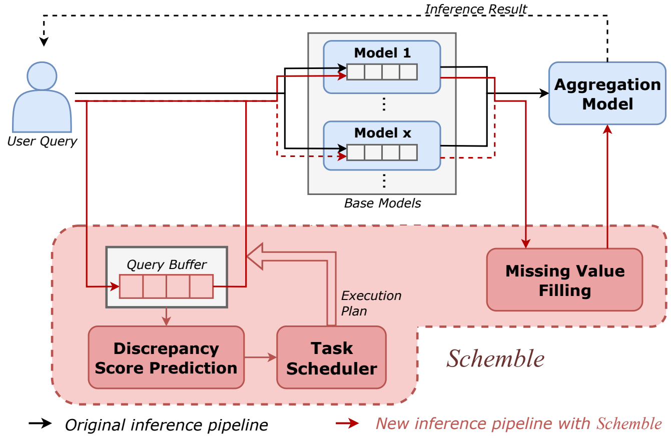


Fig. 3: The original ensemble inference pipeline(black) and the new inference pipeline with *Schemble*(red)

V. Discrepancy Score Prediction

We first define a novel metric, discrepancy score, for measuring query difficulty using base models' outputs and the ensemble's outputs. Since models' outputs are not available before the execution, we train a lightweight neural network to predict newly arrived queries' difficulty. Then we conduct offline accuracy profiling on historical data to obtain the performance of all model subsets on inputs with different hardness. This module combines the estimated discrepancy score with the profiling table to construct the utility function for subsequent inference task scheduling.

A. Discrepancy Score

There are a number of definitions of sample difficulty in the literature [27, 28]. For example, Carlini et al. [27] proposed to use the distance to the decision boundary measured by an adversarial example attack as the measurement. Jiang et al. [28] analyzed the consistency score, which characterizes the expected accuracy for a held-out instance given training sets of varying sizes. However, these metrics require either retraining multiple models or running model inference multiple times, introducing huge computation overhead to generate difficulty labels. Ensemble agreement [27] is a metric that quite fits our problem. It ranks samples' difficulty based on the agreement within the ensemble, as measured by the symmetric KL-divergence between base models' outputs. Since the inference results of historical queries could be recorded at a low cost,

[1] We write scheduling instead of selecting model subsets because we also consider task execution order

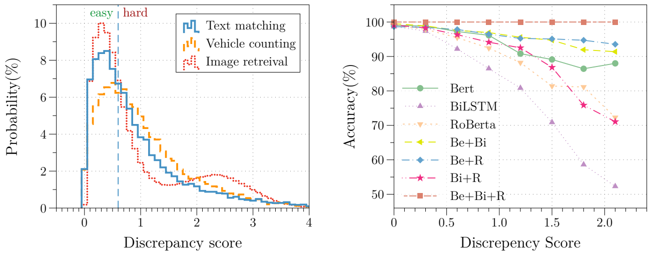
we could directly calculate the ensemble agreement score. However, problems remain in this metric.

Ensemble agreement is initially proposed for ensembles of homogeneous models. However, deep ensembles often contain heterogeneous base models whose accuracy and output distributions differ from each other. Models with lower accuracy are more likely to induce disagreement, which causes problems the ensemble agreement metric cannot address. Assume we have three binary classifiers; the first classifier is less accurate than the other two. Intuitively, a sample getting outputs '1,0,1' is more difficult than one with outputs '0,1,1'. However, the ensemble agreement metric cannot distinguish between them. Moreover, deep networks are discovered to be poorly calibrated [29], which means the outputs probability associated with the predicted class label does not align with its ground truth correctness likelihood. It thus brings in large differences between base models' output distributions and would significantly affect the KL-divergence, leading to meaningless agreement scores. To cope with these problems, we propose the discrepancy score to better measure difficulty with deep ensembles. Following the idea of the ensemble agreement, the discrepancy score of sample x is defined as follows:

$$Dis(x) = \frac{1}{m}\sum_{k=1}^{m}[Norm_x(d(f(x;\theta_k), E(x;\theta_1,...,\theta_m))] \quad (1)$$

where $f(x;\theta_k)$ is the output from the k^{th} base model and $E(x;\theta_1,...,\theta_m)$ is the ensemble's output. d is the function measuring distance between models' predictions. We use the JS-divergence to measure the distance between probabilistic distributions in classification tasks, similar to the ensemble agreement metric [27]. For regression tasks, we employ the Euclidean distance.

The expression calculates the average distance between the base models' outputs and the ensemble's outputs. Hence, it ensures base models give the same predictions as the ensemble when the score approximates zero. Under this condition, executing fewer base models for easy samples with low discrepancy scores would not incur accuracy loss. However, this may not be the case for the ensemble agreement metric, because the aggregation model may output differently even if the base models' outputs are identical. Moreover, we add normalization in the expression to eliminate the effect of different accuracy of base models. Base models with lower accuracy are more likely to give incorrect outputs and thus have larger distances to the ground truth on average. However, intuitively, getting a large distance on a low-accuracy model does not contribute to the sample's difficulty as much as getting the same distance on a high-accuracy model. As in the example mentioned above, directly summing up distances would naturally amplify low-accuracy models' impact. Hence, we normalize the distances of every base model before averaging them to diminish the contribution of inaccurate models and keep all distances at the same scale. Towards the various output distributions, we apply *temperature scaling* [29], an efficient yet effective post-processing calibration method, on classifiers' outputs. After temperature scaling, base models' distribution will be at the same scale.



(a) Distribution of the discrepancy score three real-world datasets

(b) Relation between the discrepancy score and models' accuracy

Fig. 4: Experimental analysis on discrepancy score

B. Analyze Discrepancy Score

From the definition, samples with larger discrepancy scores get more diverse outputs from the base models. Executing fewer models for these hard samples is more likely to get results that differ from the original ensemble. Thus, more models should be executed for them to keep high accuracy. On the contrary, easy samples with smaller discrepancy scores get similar outputs from the base models. Choosing fewer models or even a single model for them could get results the same as the ensemble. The discrepancy score accords with our intuition that easy samples require less computation.

To further demonstrate, fig. 4 displays the distribution of the discrepancy scores of samples from three real-world datasets. The results show that a great proportion of samples possess a low discrepancy score around zeros. We then divide the dataset into multiple bins by discrepancy scores and calculate the accuracy of different combinations of base models in each bin to see the relation between the discrepancy score and the model combinations' accuracy. Note that since this paper focuses on the efficiency of ensemble inference, we refer to the ensemble's outputs as the ground truth. Fig. 4b presents the results of the text matching task. We could observe that easy samples get high accuracy on all model combinations(over 90%), while hard samples show larger error rates with small model sets. The observation implies that huge redundancy could be eliminated with minor accuracy loss by executing fewer models for easy samples and more for hard ones. More details on experimental settings and ensemble models used could be found in Section VIII.

C. Predict Discrepancy Score

We utilize a neural network to predict discrepancy scores of newly arrived queries. We generate training data from historical inference results and define the networks to take queries as inputs and generate two outputs. The first is a prediction on the original task, and the second is the estimated discrepancy score. For example, the prediction network in text matching task outputs a matching probability and a discrepancy score prediction. We train the network with the following weighted loss function:

$$\begin{gathered} Loss(label, dis, output_1, output_2) = \\ l(label, output_1) + \lambda MSE(dis, output_2) \end{gathered} \tag{2}$$

where λ is a hyperparameter and we set 0.2 in the experiments. We regard the ensemble's output as the $label$. dis is the ground truth discrepancy score. $output_1$ and $output_2$ are outputs from the prediction network. l is a proper loss function for the original task, and MSE is the mean square error. We found that adding the original task label into training could improve the prediction of discrepancy score, which we believe is because sample difficulty is closely related to what we expect to derive from the sample. Note that $output_1$ will not be used during the inference phase.

Specific network architectures are designed according to the task and the input modality. We develop two feature networks to predict discrepancy scores for two modalities: text and images. Our primary consideration in designing these networks is time efficiency since the extra computation could be critical. For text inputs, we convert text to vectors using pretrained word2vec [30] and adopt an efficient text matching model MV-LSTM [31] to generate the matching label. We concatenate the final outputs with intermediate outputs from the LSTM layer and use a densely connected layer to map them to the discrepancy score. As for images, we employ a pretrained MobileNet [32] as the feature extractor and finetune the model on specific tasks. An additional densely connected layer is trained to map the final outputs and the intermediate outputs to the discrepancy score. Outputs from the last convolutional layer are used as the intermediate outputs.

Training a neural network to predict scores for ensemble selection is similar to the Gating method [21] which also trains a neural network. Hence, we illustrate why predicting discrepancy scores is more applicable than the Gating method by experimental analysis in this part. The main difference between our method and the gating is that we apply neural networks to learn queries difficulty while gating aims to learn models' preferences. The gating network takes the query as the input and outputs a weight for each model indicating how accurate that model will be, which is equivalent to learning whether each model is correct on the current query. Notice that the discrepancy score computes the average distances between all base models' outputs and the ensemble's outputs. Take the ensemble's outputs as the ground truth; if the distance between the base model θ_k's output and the ensemble's is small enough, the base model θ_k must be correct. Therefore, in some sense, the gating network is trying to estimate whether $d(f(x;\theta_k), E(x;\theta_1, ..., \theta_m))$ is large for every k.

As previous works [33] discovered, deep models suffer from high variance. Random initialization could result in diverse model weights even with the same architecture and data. Thus the preference space of deep models is often complicated and hard to capture. We conduct experiments on a public image dataset CIFAR100 [34] to demonstrate our points. We apply six popular image classification models and average them to obtain an ensemble. Fig. 5 shows the correlation coefficients between models' preferences with different architectures and random seeds. Preference of model θ_k refers to the vector $[d(f(x_1;\theta_k), E(x_1)), ..., d(f(x_n;\theta_k), E(x_n))]$. The discrepancy score computed from the ensemble is added to the table for comparison. We observe that the preferences of models are poorly consistent between different architectures and random seeds, indicating that preference space is random and complicated. Moreover, Gating have to learn preferences of all base models to conduct selection, adding another layer of difficulty to the learning task. Conversely, the discrepancy score shows a much stronger correlation under different random seeds. It gets a relatively stable ranking between samples. Plus, we only predict one score representing the difficulty of the sample instead of the complicated preferences of multiple models. Therefore, we believe that predicting discrepancy scores requires a smaller model capacity than learning models' preferences and thus better solves our problem.

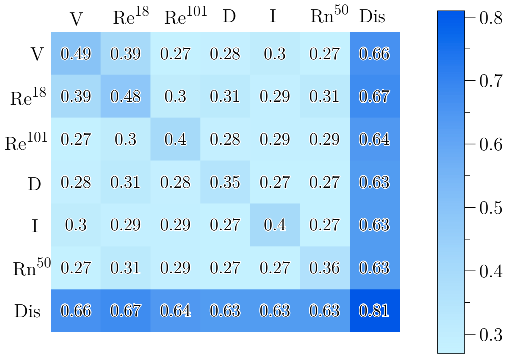


Fig. 5: Correlation coefficients between models' preferences and the discrepancy scores. Diagonal scores between an architecture and itself show the correlation of preferences for the same architecture trained with different random seeds. Diagonal score on the discrepancy score refers to the correlation of ensemble models trained with different seeds. V, Re^{18}, Re^{101}, D, I, Rn^{50}, Dis stand for VGG16, Resnet18, Resnet101, Densenet121 [35], Inceptionv3 [36], Renext50 [37] and the discrepancy scores, respectively.

D. Model Combination Accuracy Profiling

Having obtained the difficulty predictions of queries, we now build the bridge between the discrepancy score and the model inference task scheduling. Same as Section V-B, we divide the historical data into multiple bins based on the discrepancy score to profile the accuracy of different combinations of base models. Discrepancy score would primarily affect the accuracy gain as shown in Fig. 4b. With the profile, the scheduler could choose appropriate model combinations for each query according to the discrepancy score by maximizing accuracy rewards.

As mentioned earlier, the ensemble size of deep ensembles is usually small. Accuracy profiling could be performed at a low cost. However, we should also be prepared for cases where the ensemble size grows and the profiling progress brings huge computation overhead. In such a case, we profile accuracy for model combinations of size less than three and estimate accuracy for other combinations by the marginal reward function. We denote the accuracy of model combination $\{m_1, ..., m_k\}$ on the i^{th} bin as $U(i, \{m_1, ..., m_k\})$. We apply the following expression to estimate the accuracy.

$$\begin{aligned} U(i, \{m_1, ..., m_{k+1}\}) &= U(i, \{m_1, ..., m_k\}) \\ +\gamma_k \frac{1}{k} \sum_{q=1}^{k} &[U(i, \{m_q, m_{k+1}\}) - U(i, \{m_q\})] \end{aligned} \tag{3}$$

Models are sorted by their accuracy, and γ_k is the diminishing factor of marginal reward that could be estimated by profiling accuracy. We show that the estimated accuracy could approximate true accuracy nicely in experiments.

VI. Task Scheduler

Treating the ensemble inference progress as multiple base model inference tasks, this module schedules tasks (i.e select model subsets) for arrived queries based on their difficulty and decides tasks execution order. Since we execute base models in parallel, we refer to the tasks execution order as the order of inference tasks of the same model. The scheduled tasks for a query constitute the execution of a small ensemble. The goal is to ensure that as many samples can be inferred accurately before their deadlines. We use the profiled accuracy as the rewards function for executing model combinations. A scheduling algorithm is designed to address the trade-off between inference accuracy and deadline misses by maximizing cumulative rewards.

A. Scheduling Problem

The problem addressed in this module is choosing a model subset for each query and scheduling the inference tasks so that the cumulative rewards are maximized. The unit of the scheduling is a base model inference task. A deadline is assigned at each query arrival, denoting the time by which the query must be processed. Deadlines appear as a constraint that queries completed after the deadlines are considered missed and incorrect, thus get 0 rewards. Deep network execution is non-preemptive, but the execution time is approximately constant. We could estimate the completion time of tasks beforehand to see whether they could meet the deadline. To be generalized, we do not pose distribution assumptions on query arrival times. Future arrival times and deadlines are unknown, which makes the scheduling problem an online scheduling problem. Mathematically:

Let $\{q_1, \ldots, q_n\}$ be the arriving queries. q_i arrives at time r_i with deadline d_i. Rewards U_{i,s_i} that q_i could gain from model sets s_i is estimated by profiling in the Discrepancy score prediction module, where $s_i \in S = \{[s^{(1)}, \ldots, s^{(m)} | s^k \in \{0,1\}]\}$ is an indicator vector for model set. $s_i^k = 1$ indicates that model set s_i contains model k. m is the total number of base models. The schedule decides two outputs: model inference tasks for each query and the tasks execution order. We use $x(i)$ to denote the model set scheduled for q_i. To keep the expression simple, explicit expression of execution order is omitted and we use C_i^x to denote completion time of q_i. The mathematical formulation of the optimization problem is:

$$\underset{x}{max} \sum_i 1_{C_i^x < d_j} U_{i,x(i)} \tag{4}$$

The indicator function ensures that only queries processed by the deadline get the rewards. We assume that the utility function U_{i,s_i} satisfies the law of diminishing marginal utility.

Assumption 1 (Diminishing marginal utility). $U_{i,s'} - U_{i,s} \geq U_{i,s''} - U_{i,s'} \geq 0$, *where* s' *and* s'' *contain one more model than* s *and* s' *respectively.*

The assumption is consistent with human intuition and experiment results, under which we could infer that $U_{i,x} \geq \sum_k s^{(k)}/m$, since the maximum utility is 1.

B. Scheduling Algorithm

The scheduling problem is similar to the knapsack problem [38], especially the deterministic non-linear equality fractional knapsack (NEFK)[39, 40] problem. They both aim to maximize a non-linear utility function with resource constraints. Specifically, we optimize the non-linear rewards function constructed in accuracy profiling under constraints on query completion time. However, our deadline constraints are related to the scheduling order and completion time of multiple models. We have to consider the execution time of models chosen for the query as well as inference tasks assigned ahead of it to calculate the completion time of queires. Hence, it's intractable to formulate our deadline constraints as linear inequalities (or equalities) as in the NEFK problem. Therefore, we design a dynamic programming-based algorithm to solve the problem and express its competitive ratio relative to a clairvoyant scheduler (scheduler that accesses information of all future queries).

As the query traffic is unpredictable, we simplify the online scheduling problem to multiple local subproblems which is scheduling tasks for samples in the query buffer at each moment. The mathematical form of the subproblem is the same as the online problem (4), but with the restriction of arrival time removed since we only focus on queries that have already arrived.

To begin with, we assume model sets for queries are already chosen (i.e., tasks are fixed) and decide the execution order of the inference tasks. Since base models maintain their own task buffer, there is no strict restriction on the execution order of two queries (e.g., model A processes query 1 ahead of query 2, while model B processes query 2 first). This fact significantly increases the search space of the scheduling order. However, we prove that adding a restriction on the queries execution order does not degrade the scheduling performance (see detailed proofs in [41]).

Theorem 1. *For every schedule x, there exists a schedule that has consistent query order achieving the same or better performance.*

By forcing queries to be processed in a strict order, we only need to determine the processing order of queries. However, scheduling with deadlines and rewards is still an NP-hard problem [42]. Luckily, we could prove that Earliest Deadline First (EDF) [43] is the best schedule policy under the assumption that the model subsets for queries are fixed and it is possible to schedule them without deadline misses (the scheduled model subsets are feasible).

Theorem 2. *If inference tasks for all queries are fixed and the scheduled tasks are feasible, Earliest Deadline First is the optimal schedule policy.*

The proof is omitted due to length limits; detailed proofs could be found in [41]. Therefore, we take EDF as the underlying execution order and apply a dynamic programming-based algorithm to solve the local subproblem.

Algorithm 1 Dynamic Programming Scheduling Algorithm

```
Input: Base models inference time $\{T_k\}_{k=1,...,m}$; Available
    task set $\{q_i\}_{i=1...N}$; Utility matrix $\{U_{i,s}\}_{i=1,...,N;s \in S}$;
    Base models' remained execution time $\{t_k^{(0)}\}_{k=1...m}$
Output: Near optimal Scheduling plan $Comb_{N,u^*}$; Near op-
    timal utilities $u^*$;
for $i \in [1, ..., N]$ do
    Initialize $Comb_{i,0}$ as empty list (not choosing any model)
    Initialize $Time_{i,0}$ as base models remained execution time
end for
for $i \in [1, ..., N]$ do
    for $u \in [\delta, 2\delta, ..., i^{(\delta)}]$ do
        for $s \in S$ do
            $u' \leftarrow u - U_{i,s}$
            $T_s \leftarrow$ Inference time of combination s
            if $Time_{i-1,u'} + T_s$ meet the deadline then
                Combine s with elements in $Comb_{i-1,u'}$ and insert them to $Comb_{i,u}$
                Insert $Time_{i-1,u'} + T_s$ to $Time_{i,u}$
            end if
        end for
        Prune redundant solutions in $Comb_{i,u}$ based on $Time_{i,u}$
    end for
end for
return $Comb_{N,u}, \forall u \in [\delta, 2\delta, ..., N^{(\delta)}]$;
```

The algorithm is presented in Alg. 1. Initially, we quantize the reward function to shrink the problem size significantly [44]. All rewards $U_{i,x}$ are measured in multiples of some basic increment δ. We denote $\lfloor U_{i,x}/\delta \rfloor \delta$ as $U_{i,x}^{(\delta)}$ for simplicity. Since we plan to process queries in EDF order, we index queries in the queue according to their deadlines d_i, such that $d_1 \leq d_2 \leq \ldots \leq d_N$. The algorithm would output a model set s_i for each query and ensures that their inference tasks are completed before the deadlines. Dynamic programming formulation maintains a two-dimensional table, where one dimension represents the number of considered tasks, $i \in 1, \ldots, N$, and the other represents the quantized total rewards, $u \in \{\delta, 2\delta, \ldots, N^{(\delta)}\}$. Each cell contains the solutions to a sub-subproblem, that is, selecting model sets for the first i tasks that could attain exactly u reward in total. We use $Comb_{i,u}$ to denote the solutions in cell (i, u). For every i, the algorithm iterates through all possible model sets and updates the corresponding cells. $Time_{i,u}$ will record solutions' execution time to ensure the current schedule meets deadlines in recursions. A pruning operation is involved in every iteration to save computation and keep solutions in every cell optimal. More specifically, $Comb_{i,u}$ will be pruned by deleting solutions whose time cost on all models is greater than another solution. The best schedule plan could be determined by visiting cell $Comb_{N,u^*}$ that is non-empty with the largest u^*. Since we quantize the rewards to facilitate scheduling, the dynamic programming algorithm is not optimal for the unquantized local scheduling problem, but it achieves a near-optimal solution. Similar to [44], we could proof that the above algorithm is a $(1 - \epsilon)$ approximation of the optimal one, defining $\delta = \frac{\epsilon}{mN}$ (see detailed proofs in [41]).

Theorem 3. *Alg. 1 is a $1 - \epsilon$ approximation of the optimal local scheduling with $\delta = \epsilon/N$.*

We could derive an online scheduling algorithm with Alg. 1 solving the local scheduling problem at each moment. We proceed to show that it has a good competitive ratio relative to the clairvoyant scheduler, following the proof in [45] (see detailed proofs in [41]). The clairvoyant scheduler is defined as the optimal scheduler with complete knowledge of future arrival and maximizing the total rewards.

Theorem 4. *The online scheduling algorithm that applies Alg. 1 to solve local subproblems is 2m-competitive.*

VII. Filling Missing Value

Previous modules select an appropriate subset for every query and determine the execution order. Lastly, we illustrate how we handle the missing model outputs after subsets selection. The goal of filling is to ensure the aggregation model works properly even if some models' outputs are missing. We do not want the missing part to affect the aggregation of outputs we have gained. To achieve that, we provide three methods to cope with the missing values; each corresponds to an aggregation method.

- (Weighted) Voting: Base models' outputs vote for the final decision. We minimize the impact of the missing values by simply keeping them out of the vote.
- (Weighted) Averaging: Base models' outputs are multiplied by weights and summed to produce the final decision. We set the weights of the missing outputs to 0 and reweight the valid outputs.
- Stacking: base models' outputs are input to a meta-classifier, which is trained to aggregate decisions. Since there are no restrictions on the architecture of the meta-classifier, we need to fill the missing values with valid numbers to ensure the classifier works. We apply KNN

to find the most similar output files from a set of full inference results of historical data. We fill up the missing value with the weighted average of the model outputs of the top k most similar output files, using their distances to the target as the weights.

There are several advantages of using KNN for filling: KNN can fill multiple missing values simultaneously with low computation cost; KNN makes no assumptions on the distribution of models' outputs and, therefore, can fit in different tasks. However, the k-value can only be chosen using human experience. Our experiments show that *Schemble* is robust to varied k parameters.

VIII. Experimental Evaluation

Using three real-life datasets which cover text and image modalities, we conduct experiments to evaluate the effectiveness and efficiency of our algorithms for text matching, vehicle counting and image retrieval tasks.

Experiment Setting. We use the following setting.

Datasets. We use three real-life datasets, correspond to three different tasks. (1) *Text matching:* The text matching dataset is collected from a global bank's intelligent Q&A system. The service system receives customers' question and returns answers by matching the questions with candidates in the database. We conduct experiments with the ensemble deployed in the real-world system, consisting of three deep text matching models(Bert [46], Roberta [47], BiLSTM [48]) and an aggregation model(XGBoost [5]). The performance of the deep ensembles and the base models are shown in fig. 1b. We record a one-day query trace that contains more than 130,000 queries to simulate the traffic. Each sample consists of two questions, one from the customer and the other from the database, and a label representing whether the two questions map to the same answer. (2) *Vehicle counting:* UA-DETRAC [49], which is a real-world video dataset consisting of 10 hours of videos captured at 24 different locations, is used. Since ensembling of object detection models has been shown effective [50], we deployed three object detection models(EfficientDet-0 [51], Yolov5l6[2], YoloX [52]) to count the vehicles in video frames. The weighted average is used as the aggregation model. (3) *Image retrieval:* We apply the R1M [53], which contains more than 1 million images annotated with labels representing landmarks, as the database. We use the DELG [54] model with two different architectures to construct the deep ensemble. The weighted average is used as the aggregation model. The performance of the deep ensembles and the base models involved could be found in [41].

Algorithms. We implement *Schemble* and compare it with the following baseline algorithms.

- Original: Original inference pipeline that executes all base models on every query.
- Static: Statically a subset of models and deploy replicas of chosen models to utilize resources. We choose the best subset by greedy search since the ensemble size is small.
- Dynamic Ensemble Selection(DES): We apply the SOTA DES algorithm Fire-DES++ [20] that dynamically selects models for queries based on its features. For a fair comparison, a pretrained feature extractor is applied to extract low-level features from high-dimensional inputs.
- Gating: We select models by thresholding the gating weights. The same network architecture as the discrepancy score prediction model in *Schemble* was applied as the gating network.
- *Schemble(ea)*: The proposed Framework using the ensemble agreement as the difficulty measurement.
- *Schemble*: The proposed Framework.

Query traffic and evaluation metric. Deep ensembles should respond to stochastic and bursty query traffic. For the text matching task, we use the recorded one-day query trace to evaluate our framework's effectiveness towards bursty traffics. As shown in Fig. 1a, the peak arrival rate is more than 30x the troughs. Deadlines are set as constant since we treat all customers the same. For the vehicle counting task, we simulate query traffic by the Poisson process with a constant arrival rate. Each query arrives with a predefined deadline associated with its cameras to simulate locations with different priorities. Deadlines for each camera are sampled randomly from the uniform distribution. For the image retrieval task, we simulate query traffic by the Poisson process with a constant arrival rate and assign constant deadlines to all queries.

We consider a query to miss its deadline if the scheduler fails to run any model inference task for it by the deadline. Otherwise, we consider that the query is processed and return a valid result. We compute the accuracy and deadline miss rate for different baselines. Queries that miss their deadline are considered incorrect. We refer to results from the original deep ensemble as the ground truth since our focus is not on improving accuracy but on optimizing the resource efficiency of the ensemble with trusted accuracy.

Implementation. We deploy the algorithms on a machine powered by an NVIDIA Tesla P100 GPU, a 12-core Xeon E5-2650 v4 CPU, and 32 GB memory. *Schemble* is implemented in Python. All deep learning models are built using either Tensorflow[3] or Pytorch[4]. All model inference tasks are executed using GPU, and queries are sent to the server through RPC.

Baseline models are implemented with separate inference interfaces and have their own tasks buffer. If a query arrives when there are idling models, the scheduler would directly assign the query to the idling model without running the DP scheduling algorithm. Otherwise, the query will wait in the extra query buffer described in Section IV. The Discrepancy Score Prediction module deployed on GPU processes queries

[2] https://github.com/ultralytics/yolov5

[3] https://www.tensorflow.org

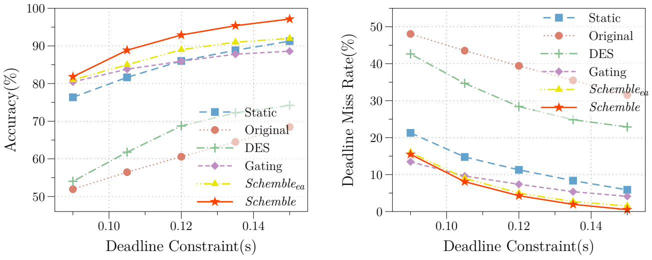
[4] https://pytorch.org

in the extra query buffer and predicts their discrepancy scores. We store the estimated discrepancy scores with the query. The task scheduler then collects queue status from base models' task buffers and takes query information(discrepancy scores, deadlines, arrival time) from the query buffer as the inputs. The scheduling algorithm selects model sets for all queries and determines which query to process next. We execute the algorithm again whenever there is a new arrival to produce a better execution plan. After the execution, the scheduler won't assign selected inference tasks to the corresponding models' immediately. Instead, it waits until there are idling models, because there may be new arrival during the waiting time. Waiting for the possible arrival is beneficial to an efficient execution plan. There will be no extra waiting time since the tasks will be assigned to models as soon as there are idling ones. We execute the scheduling process on the CPU as scheduling within GPU is pretty restrictive.

Experiment Results. We next report our findings. For more experiments, please refer to the full version [41].

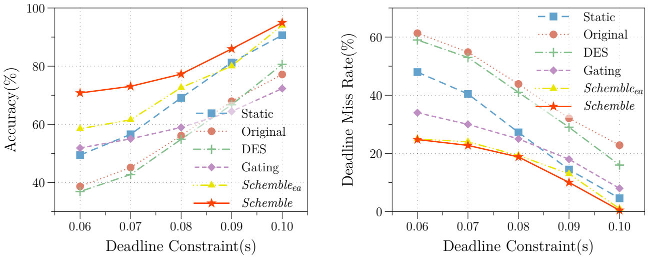
Exp-1: Overall Performance. We first evaluate *Schemble* in terms of accuracy and deadline miss rate on three tasks. We manually change the deadline constraint to simulate different deadline requirements. Worth mentioning that all deadlines assigned are larger than the time delay of the slowest model in the ensemble, in which case deadline misses will only result from queue blocking. We allow rejection for all baselines, which means queries that have an estimated completion time exceeding their deadlines would be rejected.

The evaluation results are presented in Fig. 6, Fig. 7 and Fig. 8. Under a deadline constraint of 120ms, *Schemble* increases the accuracy by 32.9% and reduces the deadline miss rate by 35% compared to the original pipeline on the text matching task. Tab. I shows the average accuracy and deadline miss rate under different deadline constraints. *Schemble* clearly outperforms other methods on all datasets, reducing the deadline miss rate significantly and achieving the highest accuracy. Note that the accuracy and the deadline miss rate show opposite patterns because we treat the missed queries as incorrect samples. However, minor differences still exist because the accuracy of the processed queries between baselines differs. The improvement of *Schemble* comes from two reasons: 1) dynamic model execution allows more flexible scheduling. *Schemble* chooses model set according to the current queuing status to effectively lower the deadline miss rate, while other baselines execute models merely based on input features or offline accuracy profile. The static selection shows comparable performance with Gating and better accuracy than DES. We observe that Gating and DES select identical model sets for most queries, which induces a waste of resources. Gating often executes the fastest model, reducing the miss rate but having low accuracy; DES tends to execute the model with the highest accuracy, producing accurate results, but the miss rate is high. 2) *Schemble* treats inputs differently based on the estimated discrepancy score. As analyzed in Section V-B, our design explores and harnesses inputs' difficulty, while other baselines focus more on which model could output the correct result. Their approach cannot distinguish between inputs in the deep ensemble case because of the high dimension of the input space. We also add the *Schemble(ea)* baseline to see how our novel difficulty measurement outperforms the existing approach. *Schemble(ea)* achieves comparable deadline miss rate since it also employ the task scheduler. However, it gets a lower accuracy since the ensemble agreement metric could not effectively distinguish inputs.



(a) Accuracy (b) Deadline Miss Rate

Fig. 6: Comparison between baselines on text matching under different deadlines

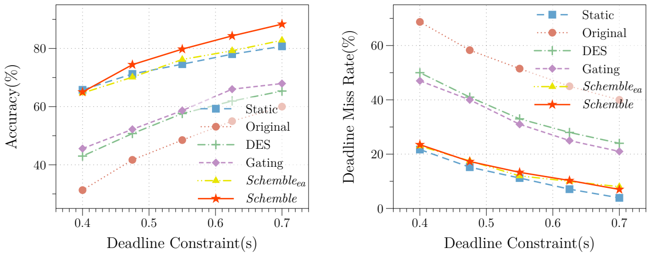


(a) Accuracy (b) Deadline Miss Rate

Fig. 7: Comparison between baselines on vehicle counting under random deadlines with different means.

To inspect how *Schemble* reacts to bursty traffic in the intelligent Q&A system, Fig. 9 displays the average accuracy and deadline miss rate in different time segments. *Schemble*, Static and Gating achieve miss rates close to 0 in 0-8h where the query traffic is light. *Schemble* gets better accuracy through difficulty-dependent scheduling, utilizing three models for inference. There is a noticeable accuracy loss for Static since it only deploys two models. It got worse for Gating because it often selects the fastest model with the lowest accuracy to ensure efficiency. DES and Gating select model sets regardless of the queue status. Thus their selections remain unchanged even when the traffic has multiplied by 30 times. During 10-18h, where queries arrived much faster, all methods' miss rates increase. *Schemble* increases the least by scheduling fewer models adaptively.

Exp-2: Latency. When evaluating the deadline miss rate, we allow rejection for all baselines. We skip the query when its estimated completion time exceeds the deadline. However, all queries have to be processed eventually in some scenarios. We force every query to be processed and report the latency and accuracy in Tab. II. *Schemble* achieves 500x



(a) Accuracy

(b) Deadline Miss Rate

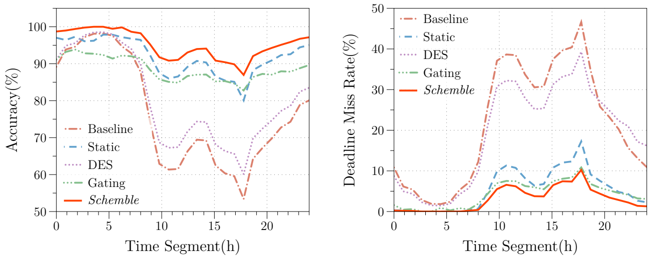
Fig. 8: Comparison between baselines on image retrieval under different deadlines.

TABLE I: Comparison of average Accuracy(Acc) and Deadline Miss Rate(DMR) performance among different deadline constraints between baselines on two tasks

	TM		VC		IR	
	Acc	**DMR**	**Acc**	**DMR**	**mAP**	**DMR**
Original	60.4	39.6	57.0	43.0	47.3	52.7
Static	84.8	12.3	69.4	26.9	74.1	**11.8**
DES	66.2	30.7	56.4	39.6	55.7	35.2
Gating	85.3	8.0	60.5	23.0	58.1	32.8
Schemble(ea)	87.6	6.8	73.3	16.3	75.0	14.5
Schemble	**91.2**	**6.1**	**80.4**	**15.4**	**78.4**	14.3

lower mean latency than the original pipeline while keeping over 97% accuracy on text matching task. Compared with other baselines, *Schemble* obtained the lowest P95 latency and max latency. Though Gating gains the smallest mean latency, it results in a larger accuracy loss. *Schemble* gets less significant improvement on latency in the other two tasks, which is because the simulated traffic is stabler and the ensemble size of the image retrieval task is smaller. In this part of the experiments, *Schemble* aims to achieve a better tradeoff between the accuracy and the latency instead of achieving the best performance on both metrics. To show that our results are significant, we display the results with two metrics as the axis in Fig. 12 and define objective function $c = 100 * Acc - \lambda * Latency$ as the weighted sum of the latency and the accuracy. The weight λ imply the relative importance of the latency metric. A larger objective indicates a better tradeoff. The curves ($Acc = \frac{\lambda}{100} Latency + \frac{1}{100} c$) in Fig. 12 show objectives functions with two different weights. Dots on the same curve achieve the same objective and moving the curves upwards increase the objectives. We could observe that our method achieves the best trade-off between all baselines under an extensive range of weights ($[0.056, 210]$). Only when the weights go extreme ($(-\infty, 0.056) \cap (210, \infty)$), the case that customer almost just care about one metric, our method achieves the second best performance.

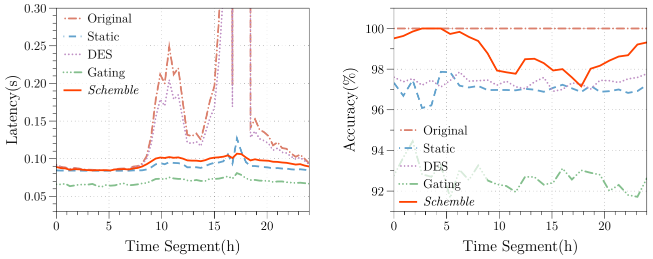
Fig. 11 shows how latency and accuracy vary in different time segments. *Schemble*, Static and Gating successfully eliminate the latency burst while Gating gets an even lower latency. We could see from Fig. 10b that compared methods show a stable accuracy when the load changes. *Schemble* better adapts to the varying traffic by taking queue status into consideration. When the traffic is light, *Schemble* utilizes three models for inference. It selects fewer models to keep the latency under control during the bursty period.



(a) Accuracy

(b) Deadline Miss Rate

Fig. 9: Comparison between baselines on one-day query traffic of text matching task



(a) Latency

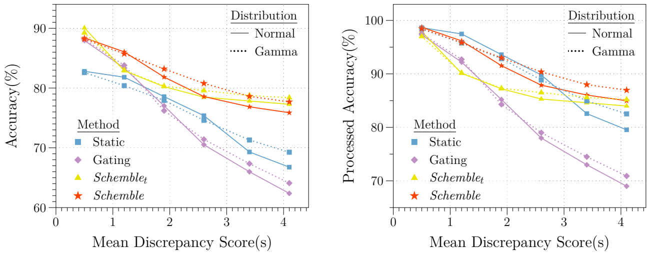
(b) Accuracy

Fig. 10: Latency and accuracy on one-day query traffic of text matching task

Exp-3: Distribution of Discrepancy Scores The distribution of the queries' discrepancy scores is closely related to the performance of our framework. We sample the queries randomly from the real-world datasets in previous experiments. In this part of the experiments, we sample data based on their true discrepancy scores and observe how different distributions affect baselines' performance. A new baseline *Schemble(t)* is considered, which is *Schemble* without the discrepancy score prediction module, to see the contribution of the first module. *Schemble(t)* assign all queries with identical discrepancy scores. We alter the distribution of queries' discrepancy scores to be Normal distribution and Gamma distribution with different means; standard variance and scale are set to 0.03 and 1, respectively. We fix deadline constraints to be 105ms and present the accuracy and the processed accuracy (ignore the missed queries) in Fig. 11a and Fig. 11b. We do not present the deadline miss rates because they practically remain the same under different distributions. Results show an obvious decreasing trend since harder examples get lower accuracy. *Schemble* outperform all other methods in terms of accuracy except *Schemble(t)* and achieve the second highest processed accuracy. We observe that as the mean of distribution grows, *Schemble* gets a larger improvement, which is counter-intuitive since harder queries leads to lesser optimization opportunity. We believe it is benefited from the task scheduler which can adaptively execute more models when there are extra resources, while other methods select regardless of the queue. Interestingly, *Schemble(t)* achieves slightly lower deadline miss rate than *Schemble* because it remove the predictor. We

TABLE II: Comparison of Accuracy(Acc) and various Latency metrics(Mean, P95, Max) between baselines on two tasks

	Text matching				Vehicle Counting				Image retrieval			
	Acc(%)	**Latency(s)**			**Acc(%)**	**Latency(s)**			**mAP(%)**	**Latency(s)**		
		Mean	P95	Max		Mean	P95	Max		Mean	P95	Max
Original	**100.0**	50.5	97.7	99.5	**100.0**	8.4	20.9	22.5	**100.0**	104.9	195.3	205.1
Static	96.9	0.11	0.23	0.64	95.0	0.12	0.31	0.90	85.0	**0.57**	**1.33**	**2.24**
DES	96.9	8.2	24.5	34.2	95.5	3.5	7.2	12.5	86.8	3.50	20.1	28.0
Gating	93.0	**0.08**	0.15	0.49	90.0	**0.11**	0.33	0.67	85.3	0.60	1.69	4.56
Schemble(ea)	96.5	0.13	0.17	0.37	94.7	0.17	0.33	0.44	89.3	0.69	1.73	2.59
Schemble	97.2	0.10	**0.14**	**0.33**	96.2	0.15	**0.28**	**0.43**	91.2	0.65	1.58	2.35



(a) Accuracy

(b) Processed Accuracy

Fig. 11: Accuracy and Processed Accuracy of baselines under Normal distribution and Gamma distribution with various means

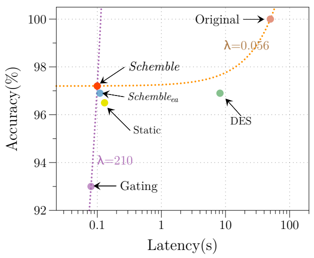


Fig. 12: Latency and accuracy of different baselines with objective function with different weights.

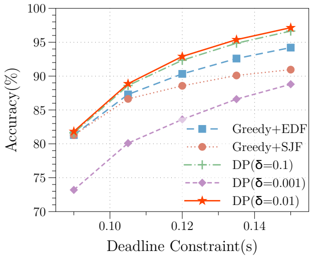


Fig. 13: Accuracy comparison between scheduling algorithms on text matching under different deadlines

observe that when the mean of the distribution is extreme (i.e the queries are all very easy or very hard), *Schemble(t)* gets a comparable processed accuracy with *Schemble* because it's hard to distinguish between queries. Thus *Schemble(t)* achieves a higher accuracy. When the mean of distribution is in 1 to 3, *Schemble* could better explore difference between queries difficulty and gets better accuracy. More results applying uniform distribution and normal distribution with larger variance could be found in [41].

Exp-4: Task Scheduler. *Schemble* applies a dynamic programming-based algorithm to solve the online scheduling problem. This part compares the proposed scheduling algorithm with several naive and traditional approaches. Besides, as mentioned in Section VI-B, we can address different trade-offs by tuning the quantization step δ in Alg. 1. We display the performance of Alg. 1 with different hyperparameter δ. With the first module fixed, the following scheduling algorithms are compared.

- Greedy+EDF/FIFO/SJF: Take EDF/FIFO/SJF as the execution order. Greedily select the model set with the highest rewards that could complete by the deadline for every query. EDF executes the query with the earliest deadline first; FIFO processes the query with the earliest arrival time first; SJF runs the query with the smallest estimated discrepancy score first.
- No selection + EDF: Take EDF as the execution order and execute all base models.
- $DP_{\delta=0.1/0.01/0.001}$: The purposed scheduling algorithm with different quantization step δ.

Results are shown in Fig. 13 and Fig. **??**. DP algorithm with $\delta = 0.01$ achieves the best performance. Our method shows little improvement when the deadline constraint is small. As it increases, DP gets higher accuracy and lower miss rate than Greedy algorithms with different execution orders since there is a larger room for task scheduling. Greedy algorithms select the model set with the highest rewards, regardless of the remaining samples in the queue. Therefore, it will incur deadline misses more easily when queries arrive quickly. By increasing δ to 0.1, DP gets more efficient but provides a scheduling plan with smaller cumulative rewards; thus a slight drop in accuracy occurs. Theoretically, Alg. 1 could output results closer to the optimal plan when we reduce δ to 0.001. However, the execution time of the Dynamic programming algorithm will increase linearly as the table size is multiplied by 10. With the extra time affecting the inference, DP($\delta = 0.001$) shows a high deadline miss rate and thus low accuracy.

Exp-5: Computation and Resource efficiency. *Schemble* introduces extra computation and memory cost in two parts, the neural network in discrepancy score prediction and the DP algorithm for task scheduling. All experiments presented above have taken the additional overhead into account. This part presents the overhead of our scheduling framework separately.

We test the latency and memory footprint of the discrepancy prediction network on GPU to compare with the deep ensemble. Fig. 14 shows that the extra network cost only 6.5% runtime and 0.4-2% memory of the ensemble.

Worth mentioning that we run two modules only on queries that are stored in the query buffer, in which case discrepancy score prediction and the scheduling could be run in parallel with ensemble inference tasks. We could eliminate the effect of the extra time delay by storing the query in the buffer only when the remaining execution time of base models exceeds a threshold.

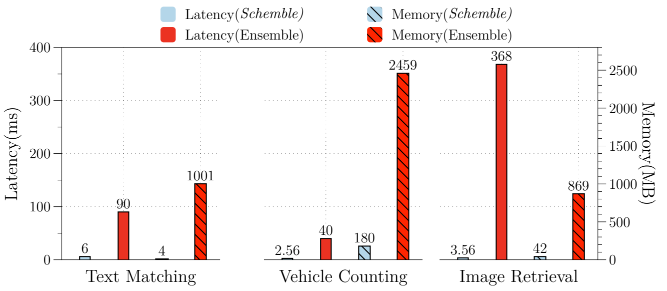


Fig. 14: Latency and memory footprint of *Schemble* and the deep ensemble on NVIDIA Tesla P100 GPU.

IX. Conclusion

In this paper, we discuss the impact of query difficulty and study the task scheduling problem for deep ensemble inference. We present *Schemble* to reduce the deadline miss rate while maintaining high accuracy. *Schemble* distinguishes between queries with different hardness through a novel difficulty measurement that adapts to heterogeneous deep ensembles. We solve the online scheduling problem with a dynamic programming-based algorithm solving local subproblems and prove that the performance of the global online algorithm is also guaranteed. Extensive evaluations on two real-world datasets show that *Schemble* could handle unpredictable traffic nicely, ensuring queries meet their deadlines and achieving better inference accuracy.

References

[1] H. Yuan, G. Li, L. Feng, J. Sun, and Y. Han, "Automatic view generation with deep learning and reinforcement learning," in *ICDE*, 2020.

[2] M. A. Ganaie, M. Hu, M. Tanveer, and P. N. Suganthan, "Ensemble deep learning: A review," *CoRR*, 2021.

[3] N. An, H. Ding, J. Yang, R. Au, and T. F. A. Ang, "Deep ensemble learning for alzheimer's disease classification," *J. Biomed. Informatics*, 2020.

[4] T. G. Dietterich *et al.*, "Ensemble learning," *The handbook of brain theory and neural networks*, 2002.

[5] T. Chen and C. Guestrin, "Xgboost: A scalable tree boosting system," in *KDD*, 2016.

[6] L. Breiman, "Random forests," *Mach. Learn.*, 2001.

[7] L. K. Hansen and P. Salamon, "Neural network ensembles," *TPAMI*, 1990.

[8] H. Suk, S. Lee, and D. Shen, "Deep ensemble learning of sparse regression models for brain disease diagnosis," *Medical Image Anal.*, 2017.

[9] X. Zhang and D. Wang, "A deep ensemble learning method for monaural speech separation," *TASLP*, 2016.

[10] L. Deng and J. C. Platt, "Ensemble deep learning for speech recognition," in *INTERSPEECH*, 2014.

[11] S. Liu, L. Liu, J. Tang, B. Yu, Y. Wang, and W. Shi, "Edge computing for autonomous driving: Opportunities and challenges," *Proc. IEEE*, 2019.

[12] S. Tuli, N. Basumatary, S. S. Gill, M. Kahani, R. C. Arya, G. S. Wander, and R. Buyya, "Healthfog: An ensemble deep learning based smart healthcare system for automatic diagnosis of heart diseases in integrated iot and fog computing environments," *FGCS*, 2020.

[13] G. Martínez-Muñoz and A. Suárez, "Pruning in ordered bagging ensembles," in *ICML*, 2006.

[14] N. Li, Y. Yu, and Z. Zhou, "Diversity regularized ensemble pruning," in *ECML/PKDD (1)*, 2012.

[15] G. Tsoumakas, I. Partalas, and I. P. Vlahavas, "An ensemble pruning primer," in *Applications of Supervised and Unsupervised Ensemble Methods*, ser. Studies in Computational Intelligence, 2009.

[16] A. H. Ko, R. Sabourin, and A. de Souza Britto Jr., "From dynamic classifier selection to dynamic ensemble selection," *Pattern Recognit.*, 2008.

[17] R. M. O. Cruz, R. Sabourin, G. D. C. Cavalcanti, and T. I. Ren, "META-DES: A dynamic ensemble selection framework using meta-learning," *CoRR*, 2018.

[18] I. Visentini, L. Snidaro, and G. L. Foresti, "Diversity-aware classifier ensemble selection via f-score," *Inf. Fusion*, 2016.

[19] R. Caruana, A. Niculescu-Mizil, G. Crew, and A. Ksikes, "Ensemble selection from libraries of models," in *ICML*, 2004.

[20] R. M. O. Cruz, D. V. R. Oliveira, G. D. C. Cavalcanti, and R. Sabourin, "FIRE-DES++: enhanced online pruning of base classifiers for dynamic ensemble selection," *Pattern Recognit.*, 2019.

[21] X. Liao, H. Li, and L. Carin, "Quadratically gated mixture of experts for incomplete data classification," in *ICML*, 2007.

[22] D. Rosenbaum and Y. Weiss, "The return of the gating network: Combining generative models and discriminative training in natural image priors," in *NIPS*, 2015.

[23] L. Xu, M. I. Jordan, and G. E. Hinton, "An alternative model for mixtures of experts," in *NIPS*, 1994.

[24] Y. Lee, A. Scolari, M. Interlandi, M. Weimer, and B.-G. Chun, "Towards high-performance prediction serving systems," *NIPS ML Systems Workshop*, 2017.

[25] D. Crankshaw, X. Wang, G. Zhou, M. J. Franklin, J. E. Gonzalez, and I. Stoica, "Clipper: A low-latency online prediction serving system," in *NSDI*, 2017.

[26] T. Woloszynski and M. Kurzynski, "A probabilistic model of classifier competence for dynamic ensemble selection," *Pattern Recognit.*, 2011.

[27] N. Carlini, U. Erlingsson, and N. Papernot, "Prototypical examples in deep learning: Metrics, characteristics, and utility," 2018.

[28] Z. Jiang, C. Zhang, K. Talwar, and M. C. Mozer, "Characterizing structural regularities of labeled data in overparameterized models," in *ICML*, 2021.

[29] C. Guo, G. Pleiss, Y. Sun, and K. Q. Weinberger, "On calibration of modern neural networks," in *ICML*, 2017.

[30] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," in *ICLR*, 2013.

[31] S. Wan, Y. Lan, J. Guo, J. Xu, L. Pang, and X. Cheng, "A deep architecture for semantic matching with multiple positional sentence representations," in *AAAI*, 2016.

[32] A. Howard, R. Pang, H. Adam, Q. V. Le, M. Sandler, B. Chen, W. Wang, L. Chen, M. Tan, G. Chu, V. Vasudevan, and Y. Zhu, "Searching for mobilenetv3," in *ICCV*, 2019.

[33] C. Shorten and T. M. Khoshgoftaar, "A survey on image data augmentation for deep learning," *J. Big Data*, 2019.

[34] A. Krizhevsky, G. Hinton *et al.*, "Learning multiple layers of features from tiny images," 2009.

[35] G. Huang, Z. Liu, and K. Q. Weinberger, "Densely connected convolutional networks," *CoRR*, 2016.

[36] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the inception architecture for computer vision," in *CVPR*, 2016.

[37] S. Xie, R. B. Girshick, P. Dollár, Z. Tu, and K. He, "Aggregated residual transformations for deep neural networks," in *CVPR*, 2017.

[38] M. Assi and R. A. Haraty, "A survey of the knapsack problem," in *ACIT*. IEEE, 2018, pp. 1–6.

[39] A. Yazidi, T. M. Jonassen, and E. Herrera-Viedma, "An aggregation approach for solving the non-linear fractional equality knapsack problem," *Expert Syst. Appl.*, vol. 110, pp. 323–334, 2018.

[40] O. Granmo and B. J. Oommen, "Optimal sampling for estimation with constrained resources using a learning automaton-based solution for the nonlinear fractional knapsack problem," *Appl. Intell.*, vol. 33, no. 1, pp. 3–20, 2010.

[41] Full version. *https://XXXXXX/Files/paper/icde2023full.pdf.*

[42] R. M. Karp, "Reducibility among combinatorial problems," in *Complexity of Computer Computations*, ser. The IBM Research Symposia Series. Plenum Press, New York, 1972, pp. 85–103.

[43] J. Xu and D. L. Parnas, "Scheduling processes with release times, deadlines, precedence, and exclusion relations," *IEEE Trans. Software Eng.*, 1990.

[44] S. Yao, Y. Hao, Y. Zhao, H. Shao, D. Liu, S. Liu, T. Wang, J. Li, and T. F. Abdelzaher, "Scheduling real-time deep learning services as imprecise computations," in *RTCSA*, 2020.

[45] S. Liu, S. Yao, X. Fu, R. Tabish, S. Yu, A. Bansal, H. Yun, L. Sha, and T. F. Abdelzaher, "On removing algorithmic priority inversion from mission-critical machine inference pipelines," in *RTSS*, 2020.

[46] J. Devlin, M. Chang, K. Lee, and K. Toutanova, "BERT: pre-training of deep bidirectional transformers for language understanding," in *NAACL-HLT*, 2019.

[47] Y. Liu, M. Ott, N. Goyal, J. Du, M. Joshi, D. Chen, O. Levy, M. Lewis, L. Zettlemoyer, and V. Stoyanov, "Roberta: A robustly optimized BERT pretraining approach," *CoRR*, 2019.

[48] A. Graves, S. Fernández, and J. Schmidhuber, "Bidirectional LSTM networks for improved phoneme classification and recognition," in *ICANN*, 2005.

[49] L. Wen, D. Du, Z. Cai, Z. Lei, M. Chang, H. Qi, J. Lim, M. Yang, and S. Lyu, "UA-DETRAC: A new benchmark and protocol for multi-object detection and tracking," *CVIU*, 2020.

[50] Á. Casado-García and J. Heras, “Ensemble methods for object detection,” in *ECAI*, 2020.

[51] M. Tan, R. Pang, and Q. V. Le, “Efficientdet: Scalable and efficient object detection,” *CoRR*, 2019.

[52] Z. Ge, S. Liu, F. Wang, Z. Li, and J. Sun, “Yolox: Exceeding yolo series in 2021,” *arXiv preprint arXiv:2107.08430*, 2021.

[53] F. Radenovic, A. Iscen, G. Tolias, Y. Avrithis, and O. Chum, “Revisiting oxford and paris: Large-scale image retrieval benchmarking,” in *CVPR*. Computer Vision Foundation / IEEE Computer Society, 2018, pp. 5706–5715.

[54] B. Cao, A. Araujo, and J. Sim, “Unifying deep local and global features for image search,” in *ECCV (20)*, ser. Lecture Notes in Computer Science, vol. 12365. Springer, 2020, pp. 726–743.

Appendix

Proof of Theorem 1. Assume that query i and query j violate the consistent order restriction; there exist two models, model A and B, that model A processes query i prior to query j while model B processes query j first. Denoting C_{iA} as completion time of model A processing query i, we have $C_{iA} < C_{jA}$ and $C_{iB} > C_{jB}$. Without loss of generality, let $C_{iB} \geq C_{jA}$. If we switch the processing order of two queries in model A, the new completion time will be $C'_{iA} = C_{jA}, C'_{jA} = C_{iA}$. Since $C_i \geq C_{iB} \geq C'_{iA}$and $C_j \geq C_{jA} \geq C'_{jA}$, the completion time of query i and j won't get larger.

Repeat the switching until all queries in the schedule possess a consistent processing order. According to the analysis above, the resulting new schedule x' does not complete any query later than schedule x. As schedule x' only modifies the order with inference tasks remaining the same, it could achieve the same or better performance. □

Proof of Theorem 2. We denote OPT as the optimal scheduling of the local scheduling problem. We have provided Alg. 1 as the optimal solution to the quantized version problem. Due to the optimality, we have

$$\sum_i 1_{C_i<d_i} U^{\delta}_{i,Alg_1(i)} \geq \sum_i 1_{C_i<d_i} U^{\delta}_{i,OPT(i)} \tag{5}$$

Since $U_{i,Alg_1(i)} \geq U^{\delta}_{i,Alg_1(i)}$, we have

$$U_{i,Alg_1(i)} \geq U^{\delta}_{i,Alg_1(i)} \geq \sum_i 1_{C_i<d_i} U^{\delta}_{i,OPT(i)} \tag{6}$$

Suppose OPT at least processes one query; then the optimal total utility is greater than $\frac{1}{m}$.From the right hand side of 5, with $\delta = \frac{\epsilon}{mN}$, we could induce that

$$\begin{aligned} \sum_i 1_{C_i<d_i} U^{\delta}_{i,OPT(i)} &\geq \delta \sum_i 1_{C_i<d_i} \big(\frac{U_{i,OPT(i)}}{\delta} - 1\big) && (7)\\ &\geq \sum_i 1_{C_i<d_i} U_{i,OPT(i)} - \frac{\epsilon}{m} && (8)\\ &\geq (1-\epsilon) U_{OPT} && (9) \end{aligned}$$

where U_{OPT} is the total utilities gain from the optimal scheduling OPT. With (6) and (9) we could complete the proof. □

Proof of Theorem 3. We refer to a task as a base model inference on one query. Processing a query comprises executing one or more tasks. We denote Q_i as the set of queries that were processed under the optimal clairvoyant scheduling algorithm(OPT). Define Q_j as the set of queries processed by Alg. 1. We apply charging arguments to complete the proof, charging the utility of Q_i to that of Q_j.

Let $Q^1_i \subseteq Q_i$ be the queries in Q_i that are also processed by Alg. 1. We charge the utility of this set of queries to the processing in Alg. 1. Since the utility function satisfies assumption 1 and the maximum utility is 1, each query in Q^1_i is charged for at most m times. $Q^2_i = Q_i \backslash Q^1_i$ is the set of queries that are not processed by Alg. 1. Assume a task scheduled by OPT for $q_k \in Q^2_i$ start from t_a and end on t_b. Then at time t_a, the corresponding model must be processing another query under Alg. 1 scheduling because Alg. 1 can schedule tasks for queries in the queue near-optimally and q_k is in the queue. We charge the utility of the task scheduled by OPT to the task scheduled by Alg. 1 that is running at the starting time of the former. By the assumption, we could charge each task at most m times. Though different models could be run in parallel, inference tasks assigned to one deployed model are executed in serial, so no two tasks in OPT would charge the same task in Alg. 1. By summing utilities of tasks, each query in Q^2_i is also charged for at most m times.

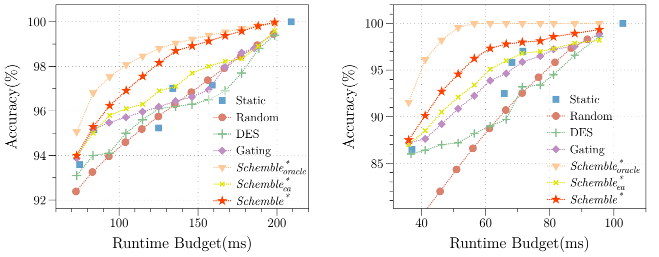
Thus, we could charge the utilities of all the queries processed under the offline optimal, with each query processed by Alg. 1 being charged no more than 2m times. □

Exp-4: Discrepancy Score Prediction. Existing works did not consider the scheduling of model inference tasks since they aim to control accumulative execution time on offline datasets instead of latency on online data streams. In this part, we compare accuracy under different cumulative execution time budgets in their favor to illustrate the effectiveness of discrepancy score prediction. Without considering online arrival, we select models for execution on offline datasets. The cumulative execution time is the sum of the runtimes of the executed models. We report the maximum accuracy under varying budgets. Extra baselines are listed below:

- Random: Randomly execute models for every sample until the budgets are met.
- *Schemble**: We estimate the discrepancy score of every sample and select model sets to maximize total rewards under the budget constraint. The optimization problem differs from the scheduling problem, so we call this approach *Schemble**. The optimization problem could be solved directly by Linear programming.
- *Schemble**(Oracle): The model sets are selected based on the ground truth discrepancy scores to show the best we could achieve with discrepancy score measurement.
- *Schemble**(ea): *Schemble** with ensemble agreement as the difficulty measurement.

Results are shown in Fig. 15. *Schemble** remarkably outperforms other baselines. Under an average runtime budget of 150ms, our framework obtained a 2x lower error rate than all other methods in the text matching task. When we have tight budgets, the accuracy of all methods is relatively close to each other since executing one model almost costs all the budgets. As the budget increases, our framework and the oracle version show a greater boost in accuracy, utilizing the limited budgets by exploiting sample difficulty. Though getting the ground truth discrepancy score is impossible, our method provides a great approximation through the prediction module.

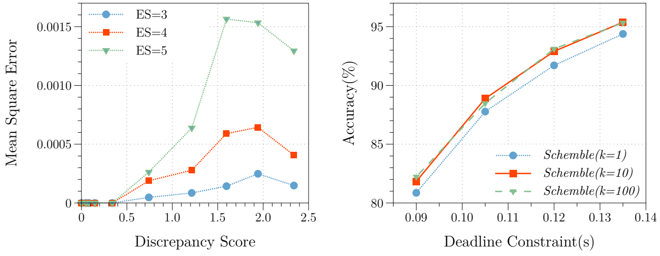
Static method runs identical model subsets for all samples; thus, only several points are generated by iterating through possible combinations, which brings obstacles to adapting to various budgets. Gating shows comparable performance with



(a) Text matching

(b) Vehical counting

Fig. 15: Comparison between baselines under different average runtime budgets on text matching and vehicle counting



(a) Mean Square Error

(b) Accuracy

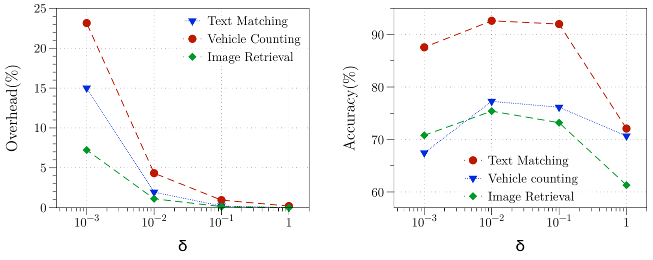
Fig. 16: Left: MSE(Mean Square Error) between the estimated accuracy and the true accuracy on samples with different discrepancy scores. ES refers to ensemble size. Right: Accuracy of *Schemble* with different k. k is the parameter in KNN.

static selection but is able to meet different budgets. With closer inspection, we find that the gating network outputs similar weights for all samples, which means it fails to discriminate between input features. This observation supports the analysis in Section V-C that learning preferences of multiple deep models are too challenging for a lightweight model.

Exp-5: Accuracy profiling and KNN. In the experiments above, we calculate the accuracy of all model combinations to obtain the utility function. *Schemble* offers an estimation method for accuracy profiling when the ensemble size grows. We train six popular image classification models and conduct experiments on the CIFAR100 dataset [34] to test estimation performance. The models used are identical to those tested in Fig. 5. Fig. 16a presents the mean square error between the estimated accuracy and the true accuracy of ensembles of different sizes. Estimated accuracy shows MSE less than 1.6e-4, proving that the estimation could perfectly approximate the factual accuracy.

We apply KNN to fill missing outputs when stacking aggregation is involved. The parameter k affects the inference accuracy. We vary k from 1 to 100 and test the *Schemble*'s performance. As shown in Fig. 16b, *Schemble* is robust to the k-value. A small k-value results in a minor loss of accuracy, while increasing k from 10 to 100 does not improve the accuracy further.

Exp-7: Hyperparameter δ For the scheduling algorithm, we investigate the impact of hyper-parameter δ on the overhead and accuracy, as illustrated in Fig. 17. δ shows similar behaviors on both tasks. According to Thm. 3, a smaller quantization step δ leads to scheduling with cumulative rewards closer to the optimal policy. However, smaller δ also results in higher time complexity of the scheduling process, using up the time for ensemble inference. Therefore, we could address different trade-offs between optimality and efficiency by tuning δ. Based on the results, we choose $\delta = 0.01$ as the best choice in practice.



(a) Overhead

(b) Accuracy

Fig. 17: Overhead and performance with different quantization step δ