SINGA-Easy: An Easy-to-Use Framework for MultiModal Analysis

Naili Xing\textsuperscript{1}, Sai Ho Yeung\textsuperscript{1}, Chenghao Cai\textsuperscript{1,4}, Teck Khim Ng\textsuperscript{1}, Wei Wang\textsuperscript{1}, Kaiyuan Yang\textsuperscript{1}, Nan Yang\textsuperscript{1}, Meihui Zhang\textsuperscript{2}, Gang Chen\textsuperscript{4}, Beng Chin Ooi\textsuperscript{1}

\textsuperscript{1}National University of Singapore, Singapore, \textsuperscript{2}Beijing Institute of Technology, China, \textsuperscript{3}Zhejiang University, China, \textsuperscript{4}National University of Singapore (Suzhou) Research Institute, China

ABSTRACT

Deep learning has achieved great success in a wide spectrum of multimedia applications such as image classification, natural language processing and multimodal data analysis. Recent years have seen the development of many deep learning frameworks that provide a high-level programming interface for users to design models, conduct training, and deploy inference. However, it remains challenging to build an efficient end-to-end multimedia application with most existing frameworks. Specifically, in terms of usability, it is demanding for non-experts to implement deep learning models, obtain the right settings for the entire machine learning pipeline, manage models and datasets, and exploit external data sources all together. Further, in terms of adaptability, elastic computation solutions are much needed as the actual serving workload fluctuates constantly, and scaling the hardware resources to handle the fluctuating workload is typically infeasible. To address these challenges, we introduce SINGA-Easy, a new deep learning framework that provides hyper-parameter tuning at the training stage, dynamic computational cost control at the inference stage, and intuitive user interactions with multimedia contents facilitated by model explanation. Our experiments on the training and deployment of multi-modality data analysis applications show that the framework is both usable and adaptable to dynamic inference loads. We implement SINGA-Easy on top of Apache SINGA and demonstrate our system with the entire machine learning life cycle.

CCS CONCEPTS

- Information systems → Multimedia information systems;
- Computing methodologies → Machine learning;
- Human-centered computing → Systems and tools for interaction design.

KEYWORDS

depth learning, data analytics, multimedia application, distributed training

1 INTRODUCTION

Deep learning has been successfully adopted in a variety of multimedia applications such as image classification, speech recognition and news recommendation. Driven by the increasing demand of real-world multimedia applications and an unprecedented growth of big data, many Deep Learning (DL) techniques and systems [40, 41] have been developed to facilitate the development of AI applications. Although human-level performance has been achieved in areas like computer vision [43], natural language processing [11] and speech processing [42], the mass adoption of AI applications remains elusive due to two major challenges.

The first challenge is usability [33]. Many AutoML frameworks have been developed to improve usability. These include Auto-WEKA [20], H2O AutoML [21], Auto-Sklearn [9], Auto-Pytorch [45], and Auto-Keras [18]. Among these frameworks, some provide functionalities for hyper-parameter tuning that can work with large datasets, and some support good user interaction and experience. However, most of them do not take into account both.

Specifically, for hyper-parameters tuning, most DL model training processes focus on searching for the best hyper-parameter configuration using a set of stochastic gradient descent based optimization algorithms [3, 4, 19]. Many AutoML systems [22, 25] use Bayesian Optimization (BO) [30, 36] to tune the hyper-parameters automatically. This is often time consuming as they need to evaluate many different combinations of hyper-parameters to obtain the best configuration. To accelerate the searching process, AI-as-a-Service platforms such as Rafiki [39] tune the hyper-parameters of the SGD algorithms in a distributed manner, but it pays little attention to the model (or architecture) related hyper-parameters.

In terms of user interaction and experience, DL systems typically hide implementation details and appear like a black-box to users. To provide better interaction and user control, an easy-to-use APIs for managing the ML job in a finer-grained manner is required. In addition, a good model explanation solution [31, 32, 34] is also much needed in real-world applications, especially for those high-stakes applications. For example, in the X-ray based medical diagnosis [10, 27, 38], making a wrong decision may lead to catastrophic consequences, and meanwhile, providing explainable AI solutions [12, 16, 37] can also engender user trust. Auto-WEKA and H2O AutoML provide graphical user interfaces for datasets, models and task management. In addition, H2O AutoML provides a number of model explanation functions based on variable importance and dependency. However, most of these existing systems...
do not provide full support for automatic hyper-parameter tuning, good user interaction and model explanation.

The second challenge is adaptability – inference services have to support elastic computation control in real-time, as it is not practical and often infeasible to scale the computational resources of the system to handling peak workloads. In practice, the users may want to obtain more prediction results within a stipulated time and meanwhile can tolerate a slight decrease in accuracy. A conventional static model takes a fixed amount of computation and thus cannot trade off accuracy and efficiency dynamically, which therefore is unable to handle the fluctuating workload. Clipper [7] focuses on the prediction serving by introducing a new framework and explores several optimization techniques such as caching and model selection to improve latency and accuracy. However, it simply drops the instances if they cannot be processed within the time limit in the presence of high workloads. Model-Switching [44] - an online scheduler on top of Clipper, can select and switch to a different serving model based on the budget dynamically, which can achieve higher effective accuracy. Nevertheless, multiple models need to be trained beforehand and loaded to memory to support runtime model selection, which incurs additional overheads. Notably, the ability to efficiently and effectively adapting the serving model size to the current workload and computational resources available is still lacking in most systems.

To address these two challenges, namely the usability and adaptability, an easy-to-use deep learning framework supporting automatic hyper-parameter selection, distributed training, dataset and model management, model explanation, and elastic computation control is required. We design and implement such a system on top of Apache SINGA [26]. The main contributions of this work are summarized as follows:

- We present an end-to-end open-sourced DL framework called SINGA-Easy, which is developed to facilitate the adoption of DL algorithms and inference services by domain-specific multimedia application users. SINGA-Easy can automatically tune training jobs for pre-built complex models and adapt the model size when facing high inference workloads. It also provides an intuitive APIs to manage the whole DL/ML life cycle and retrieve the inference result from various perspectives.
- At the training stage, we propose a new auto-tuning framework combining both a distributed hyper-parameter tuning policy and an adaptive regularization method, which reduce the effort required for training an efficient and accurate model. We also integrate to our system a novel technique to adapt the serving model size dynamically.
- At the inference stage, we focus on the evaluation metric effective accuracy and propose a new scheduling algorithm to adapt the model size to the current workload in real-time for achieving higher effective accuracy, and meanwhile satisfy user-defined response time requirement under the available computational resources. The algorithm also reduces manual effort in the deployment, scaling and workload balancing of the service.
- To facilitate the adoption of multimedia applications, we integrate commonly used algorithms and provide an easy-to-use APIs. We also provide an option for users to evaluate model performance from the model explanation perspective provided by LIME [32] and Grad-CAM [34].

- We demonstrate the usability and adaptability of our SINGA-Easy by conducting experiments on various real-world datasets and showcasing several multimedia applications.

The remainder of the paper is structured as follows. Section 2 introduces the system architecture and the dataflow between the system components. Section 3 introduces the dynamic model serving framework. Section 4 presents system usability. Section 5 presents the experimental study. We review the related work in Section 6 and conclude the paper in Section 7.

2 SYSTEM ARCHITECTURE

This section introduces the system architecture of SINGA-Easy. The software stack is illustrated in Figure 1 and Figure 2 respectively. The system consists of the frontend layer, the backend layer, and the storage layer. Specifically, the frontend layer provides different HTTP APIs to manage both data and tasks. Users can interact with the framework via either Python SDK-Client or Web UI. In the following subsections, we will introduce the other layers in detail.

2.1 Backend Layer

SINGA-Easy is built on top of the base architecture of Apache SINGA. The backend of the overall system comprises five essential components: Admin, Training Worker, Advisor, Predictor and Inference Worker.

Admin is the core component of the system’s control plane, which exposes HTTP APIs for users to manage the whole ML lifecycle. Upon receiving requests from users via RESTful APIs, it deploys a number of workers for model training and serving, and stores information of the user-defined tasks into a Metadata Store. When a worker is deployed, the worker pulls the information of the task from the Metadata Store and starts the task.

Training Worker trains models by conducting trials proposed by a corresponding Advisor Worker. The computational kernel of Training Worker supports various DL libraries in addition to Apache SINGA e.g., PyTorch [28], and TensorFlow [1]. Figure 2 shows the stack diagram with Apache SINGA as the DL framework, where the upper layers are constructed based on the lower layers. For example, the component model is defined using Layer, Autograd, Opt and Operator, etc. Each of them is built on top of...
the basic data structures of Apache SINGA such as Tensor and Communicator. We introduce the technical details of the training part in Section 2.4.

Advisor performs hyper-parameter tuning by conducting multiple trials on a training job. In each trial, it proposes the training configuration, i.e., knobs of the model and training algorithm, to be used by Training Worker. For the implementation of the Advisor, we adopt the Bayesian Optimization of Scikit-Optimize toolbox\(^1\).

Predictor is designed for ensemble modelling, which stands between users and Inference Workers. Predictor receives requests, e.g., one or many images to be classified, from the users, then forwards the requests to a number of Inference Workers and collects the prediction results. Inference Worker manages trained models for the inference jobs, which receives the request forwarded by the Predictor and performs prediction. The technical details of inference are discussed in Section 3.

2.2 Storage Layer
The storage layer contains the following components for caching and storage of data:

- **Metadata Store** is a centralized and persistent database used to store the metadata of the whole system such as user metadata, job metadata, worker metadata and model templates. We adopt PostgreSQL\(^2\) in our system.
- **In-Memory Cache** is an in-memory data structure store used for fast asynchronous communication between Training Workers and Advisor at the training stage. Redis\(^3\) is used as the in-memory data store.
- **Message Queue** is a file-based data store used for supporting asynchronous communication between Inference Workers and Predictors for Inference Jobs. Apache Kafka\(^4\) is used in our system.

2.3 Workflow
SINGA-Easy allows users to manage the whole ML life cycle and retrieve the prediction results from inference services. The user firstly uploads the model, dataset, or annotation file to Admin, which will be stored into a distributed storage (NFS), and the metadata will be stored into Metadata Store. When send use requests to start training, Admin launches one Advisor and multiple Training Workers. Advisor stores training configurations into In-Memory Cache, and Training Workers will conduct training accordingly.

\(^1\)Scikit-Optimize: https://scikit-optimize.github.io/stable
\(^2\)PostgreSQL: https://www.postgresql.org/
\(^3\)Redis: https://redis.io
\(^4\)kafka: https://kafka.apache.org

Figure 2: SINGA-Easy software stack.

In each iteration, Training Workers report the training accuracy to In-Memory Cache, which is then used by Advisor to generate new configurations for the next training iteration. After completing a training job, one Predictor and multiple Inference Workers will be created, and the user can retrieve the Predictor inference service’s URL from Admin to use the model inference services. All these components communicate with each other via a message queue.

2.4 Elastic Inference
Model slicing \cite{5} is a general technique to enable deep learning models to support elastic computation. Specifically, each layer of the model is divided into equal-sized contiguous computational groups. During both training and inference, there is a single parameter slice rate \(r\) that dynamically controls the fraction of groups involved in the computation for all the layers in the model, namely the model width. In particular, these groups are trained dynamically to build up representations residually, where the first group learns the base representation, and each subsequent group learns on top of all its preceding groups. As a result, during inference, we can support accuracy-efficiency trade-offs by dynamically slicing a subnet of a certain width, where only the parameters of the activated groups are involved in computation. Theoretically, the number of parameters and computation measured in FLOPs are both roughly quadratic to the slice rate \(r\) \cite{3}, e.g., a slice rate of 0.5 can achieve up to four times speedup. Therefore, we can support elastic inference by introducing the model slicing technique to the training stage. Specifically, we can train the model with multiple slice rates beforehand, and then at the inference stage, the model can be switched to different sub-models adaptively.

The overall training process is illustrated in Figure 3. We train the model with model slicing to render the ability of elastic computation. For efficiency, the system trains the sub-models in a distributed manner by training these sub-model instances of different slice rates in a pool of workers. After a few training iterations, all workers merge their local copy of weights and update them globally. SINGA-Easy reuses the distributed hyper-parameter tuning component of Apache SINGA and adds an adaptive Gaussian Mixture (GM) regularization technique \cite{24} to further improve the prediction performance of the model.

3 DYNAMIC MODEL SERVING
To support dynamic model serving, we further propose a scheduling algorithm based on the elastic inference enabled via model slicing. At the inference stage, the user can send multiple requests,
process a mini-batch of a fixed number of instances. Typically, the
model serving system
scheduling algorithm is formally defined as follows.

The goal of the

Briefly,
effective accuracy

can be processed by the model before the deadline
task may contains multiple instances to be processed by the serv-
queue contains the value and its deadline constraint.

Inference Workers
Elastic Model

Predictor

client

Clients

BatchSchedulerV1V2 V3 V4V5 V6

MessageQueue with a single partitionInference Worker

MessageQueue with 3 partitionsInference Worker

(a) Single model serving

(b) Multiple model serving

Figure 4: Inference stage: each instance in the message
queue contains the value and its deadline constraint.
each of which corresponds to one prediction task. A prediction
task may contains multiple instances to be processed by the serv-
model. As shown in Figure 4(a), each Client sends instances

Table 1: Summary of variables.

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>(m_i)</td>
<td>The (i)-th sub-model</td>
</tr>
<tr>
<td>(r_i)</td>
<td>Slice rate of (m_i)</td>
</tr>
<tr>
<td>(p_i)</td>
<td>Accuracy of (m_i)</td>
</tr>
<tr>
<td>(t_i)</td>
<td>Time for (m_i) to process a mini-batch</td>
</tr>
<tr>
<td>(p_i^{\text{eff}})</td>
<td>Effective accuracy of (m_i)</td>
</tr>
<tr>
<td>(K)</td>
<td>Number of trained sub-models</td>
</tr>
<tr>
<td>(n_i)</td>
<td>Number of mini-batches assigned to (m_i)</td>
</tr>
<tr>
<td>(W_i)</td>
<td>The maximum workload* that (m_i) can handle</td>
</tr>
<tr>
<td>(N)</td>
<td>Number of instances in one prediction task</td>
</tr>
<tr>
<td>(S_{mb})</td>
<td>Number of instances in a mini-batches, which is fixed during each prediction task.</td>
</tr>
<tr>
<td>(N_{mb})</td>
<td>Number of mini-batch, (N_{mb} = N / S_{mb})</td>
</tr>
<tr>
<td>(D)</td>
<td>User-defined deadline to process (N) instances</td>
</tr>
<tr>
<td>(W_{exp})</td>
<td>Expected workload of the system, (W_{exp} = N / D)</td>
</tr>
<tr>
<td>(T_i)</td>
<td>Time spent by (m_i) to process (N) instances</td>
</tr>
<tr>
<td>(T_{fast})</td>
<td>(T_i) of the fastest sub-model</td>
</tr>
<tr>
<td>(T_{slow})</td>
<td>(T_i) of the slowest sub-model</td>
</tr>
<tr>
<td>(PC_j)</td>
<td>The (j)-th scheduling policy</td>
</tr>
<tr>
<td>(PC_{best})</td>
<td>The best scheduling policy</td>
</tr>
</tbody>
</table>

*Workload: the number of instances to be processed per second.

full model (i.e., \(r_1 = 1.0\)) has the highest accuracy while the lowest
efficiency. For both scenarios, there are \(K\) sub-models trained
beforehand. The \(N\) instances are divided into \(N_{mb}\) mini-batches
each with \(N / S_{mb}\) instances by Inference Worker, and the sub-model
\(m_i\) takes \(T_i\) to process all \(N_{mb}\) mini-batches. Formally, the scheduling
algorithms are to determine the best scheduling policy denoted as \([n_1...n_m]\), where \(n_i\) is the number of mini-batches assigned to
sub-model \(m_i\).

3.1 Single Model Serving

In the first scenario where only one single model can be deployed
in the system, Algorithm 2 is adopted to dynamically adapt the
model size to meet the deadline requirement and to obtain the best
effective accuracy. The situations are summarized as follows:

- If \(D \leq T_{fast}\) (see Figure 5 where \(D \leq 4\)), dropping mini-
batch will be unavoidable. In this case, the scheduler will schedule
all mini-batches to the fastest sub-model to minimize the drop rate.
- If \(D \geq T_{slow}\) (see Figure 5 where \(D \geq 24\)), the scheduler will schedule
all mini-batches to the slowest but most accurate
sub-model.
- If \(T_{fast} < D < T_{slow}\) (see Figure 5 where \(4 < D < 24\)), more
than one sub-model is needed to achieve the best effective
accuracy.

For a single sub-model \(m_i\), it’s effective accuracy can be defined as:

\[
p_i^{\text{eff}} = \begin{cases} 
  p_i, & W_{\text{exp}} \leq W_i \\
  (W_i / W_{\text{exp}}) \cdot p_i, & W_{\text{exp}} > W_i \end{cases}
\]  

When the expected workload \(W_{\text{exp}}\) is higher than the maximal
workload that a single sub-model can handle, i.e., \(W_i\), the effective
accuracy decreases since the sub-model cannot process all
Algorithm 1 Model Serving Predictor and Inference Worker

Input: \( \mu = \{m_1, \ldots, m_K\} \), \( D \), \( N \), \( \rho = \{p_1, \ldots, p_K\} \)

Output: Prediction Result PredictionResult

1: function PREDICTOR\( (N, D) \)
2:   while True do
3:     Receive user requests
4:     Send the user requests to the message queue
5:   end while
6: end function

7: function INFEERENCE WORKER\( (\mu, \rho, D) \)
8:   while True do
9:     //Retrieve instances of same prediction task from queue
10:    instances = RetrieveInstances()
11:   //Divide instances to mini-batches
12:    for \( i = 1 \) to \( K \) do
13:      \( N_{mb} = \) DivideIntoMiniBatchList(instances)
14:    end for
15:    \( \eta, p^{eff} = \) SCHEDULER\( (\mu, \rho, D, N_{mb}) \)
16:    for \( n_i \) in \( \eta \) do
17:      Switch model to sub-model \( m_i \)
18:      PredictionResult\( _i \) ← Model\( (n_i, \gamma) \)
19:    end for
20: end while
21: return PredictionResult
22: end function

The maximization of the objective function Eq. (2) with the constraints of Eq. (3), (4), (5) and (6) can be formulated as a Integer Linear Programming (ILP) problem, which can be solved by either the classical linear programming-based Branch-and-Bound (B&B) method or Dynamic Programming (DP). Although DP can find the optimal solution in polynomial time, the solution may not be precise as \( D \) needs to be discretized, which forms a limited number of sub-problems. Thus, the B&B method is used prior to DP, which is shown in Algorithm 2. Specifically, the B&B method takes the following steps: <s1> An initial linear programming problem \( X_0 \) is constructed by grouping Eq. (2), (5), (3) and (6), and \( X_0 \) is pushed to the problem queue \( \phi \). <s2> Retrieve a linear programming problem \( X \) from \( \phi \) and get its optimal solution \( \beta \) by applying linear programming method. <s3> If all elements in \( \beta \) are integers, then \( \beta \) satisfies Eq. (4) and will be a feasible solution of the ILP problem. If \( \beta \) leads to a higher \( p^{eff} \), then \( \beta \) will be used to update \( \eta \). <s4> If \( \beta \) contains non-integers, eg. \( \beta[i] \) is a float number, then two new sub-problems are generated by merging the problem \( X \) and two respective new constraints, namely \( n_i \geq \text{int}(\beta[i]) + 1 \) and \( n_i \leq \text{int}(\beta[i]) \), which are then pushed into \( \phi \). Steps <s2>, <s3> and <s4> are iterated until \( \phi \) is empty. Finally, the optimal \( \eta \) can be obtained. If the B&B method fails to find the optimal \( \eta \), the ILP problem will be approximated as a classical 2-dimensional unbounded knapsack problem and solved by DP.

Our experiments further confirm that the optimal solution of Eq. (2) is also the best scheduling policy \( PC_{best} \) that achieves the highest effective accuracy. For example, as shown in Figure 5, when the user sets \( D = 8 \), sub-model 1 can only serve one mini-batch within the deadline. Sub-models 3 and 4 can meet the deadline but with relatively low accuracy. The best scheduling policy is a combination of sub-models 2, 3 and 4 that achieves the highest accuracy and achieve a zero-drop rate. Specifically, mini-batch 1 is assigned to sub-model 2. The model is then switched to sub-model 3 to process mini-batch 2 after processing mini-batch 1. Finally, mini-batch 3 and 4 are processed using sub-model 4.

To further save the time spent on running the scheduling algorithm, we precompute and store combinations of \( D \) and \( N_{mb} \) in In-Memory Cache to accelerate decision making.

3.2 Multiple Model Serving

In the second scenario where multiple models can be loaded to the system, the Producer will partition instances to different queues. Each queue is served by a dedicated Inference Worker. The models in other Inference Workers are replicated from the first Inference Worker. The model in each Inference Worker is able to switch between sub-models. The global effective accuracy is defined as the average of effective accuracy of each Inference Worker. Suppose there are \( b \) Inference Workers, the global effective accuracy is \( \frac{1}{b} \sum_{i=1}^{b} p^{eff}_i \). The global best effective accuracy is equal to the average of the local best effective accuracy. To get the best local effective accuracy, each Inference Worker runs Algorithm 2 separately under the original deadline constraint \( D \) and the number of mini-batches \( N_{mb} \) in the corresponding queue.
Algorithm 2 Scheduling Algorithm

Input: $\rho = \{p_1, \ldots, p_K\}, \tau = \{t_1, \ldots, t_K\}, D, N_{mb}$
Output: $\eta = \{\eta_1, \ldots, \eta_K\}$

1: function SCHEDULER($\mu, \rho, \tau, D, N_{mb}$)
2: \hspace{1em} $\eta \leftarrow [0, \ldots, 0]$
3: \hspace{1em} if $D \leq N_{mb} \times \min(\tau)$ then $\eta[\arg\min(\tau)] \leftarrow N_{mb}$
4: \hspace{1em} end if
5: \hspace{1em} if $D \geq N_{mb} \times \max(\tau)$ then $\eta[\arg\max(\tau)] \leftarrow N_{mb}$
6: \hspace{1em} end if
7: \hspace{1em} if $N_{mb} \times \min(\tau) < D < N_{mb} \times \max(\tau)$ then
8: \hspace{2em} $\phi \leftarrow \text{Queue}([\text{Eq.}(2), (3), (5), (6)])$
9: \hspace{1em} while $\phi \neq []$ do
10: \hspace{2em} $X \leftarrow \phi.Pop$
11: \hspace{2em} $\beta \leftarrow \text{Linear_Programming}(X)$
12: \hspace{2em} if $p^\phi(\beta) > p^\phi(\eta)$ then
13: \hspace{3em} if $\forall i. \beta[i] \in \mathbb{Z}$ then $\eta \leftarrow \beta$
14: \hspace{3em} else select $i$ such that $\beta[i] \notin \mathbb{Z}$
15: \hspace{3em} $\phi.Push(X \cup \{n_i \geq \text{Int}(\beta[i]) + 1\})$
16: \hspace{3em} $\phi.Push(X \cup \{n_i \leq \text{Int}(\beta[i])\})$
17: \hspace{2em} end if
18: \hspace{1em} end if
19: \hspace{1em} end while
20: \hspace{1em} if $\eta = [0, \ldots, 0]$ then
21: \hspace{2em} $\eta \leftarrow \text{2D_Unbounded_Knapsack_DP}(\rho, \tau, D, N_{mb})$
22: \hspace{1em} end if
23: \hspace{1em} end if
24: \hspace{1em} return $\eta, p^\phi(\eta)$
25: end function

client.create_dataset(name='X_ray', task='IMAGE_CLASSIFICATION', dataset_path='\X_rayTrainDataset', dependencies=[{"torch": "1.8.1", "torch_vision": "0.2.2"}])
client.create_model(name='ResNet', task='IMAGE_CLASSIFICATION', model_file_path='\ResNet.py', dependencies=[{"torch": "1.8.1", "torch_vision": "0.2.2"}])
client.create_train_job(app='X-rayImagesClassifications', task='IMAGE_CLASSIFICATION', train_dataset=train_dataset1, val_dataset=val_dataset2, budget='GHCN: 1))
modelServe = client.create_inference_job(app='X-rayImagesClassifications', app_version = '1.8.1', budget='GHCN: 1))
prediction_result = requests.post(url=modelServe.URL, data="image": "x_ray_image2.jpg", taskID=1, deadline=3))
prediction_result["model explanation"].show()
on five food datasets for food image classification and detection and visualize the results.

We use 50,000 training images and 10,000 test images from the CIFAR dataset. We use 1.2 million training images and 50,000 test images drawn from 1,000 classes in the ILSVRC 2012 dataset. We use 5,234 training images and 634 test images from the NIH Chest X-rays dataset. Each image of the X-rays dataset is classified as "healthy" or "unhealthy" and is normalized from 1600 × 1125 to 244 × 244. We also use five Singapore food datasets. The number of classes in each dataset is 55, 101, 172, 231 and 256, respectively. Each class contains 300 to 500 images of size 624 × 244.

5.1.2 Training Results. ResNet-50 is trained on each dataset with SGD. Specifically, we train 100/100/300 epochs on CIFAR-10/ImageNet-ResNet-50 is trained on each dataset with

5.2 Dynamic Model Serving Evaluation

The adaptability of SINGA-Easy is evaluated using ResNet-50 trained on dataset NIH Chest X-rays. In the following experiments, we set the mini-batch size $s_{mb}$ to 32 for the evaluation.

We first measure the inference time $t_i$ to process a single mini-batch with different sub-models. Then we measure the effective accuracy of SINGA-Easy under the first scenario where only one single model can be loaded to the system.

To measure the actual inference time, we enable GPU warm-up and GPU/CPU synchronization. We also use torch.cuda.Event to capture the time before and after model inference. Specifically, we record the inference time with different ingesting rates from 32 to 3,000 instances/second as shown in Figure 7(a). Then the inference time is averaged to obtain the $t_i$ of sub-model $m_i$. Results in Table 3 show that both the accuracy and inference time decreases with a smaller slice rate, which is consistent with the previous discussion.

To measure the effective accuracy of SINGA-Easy, we set the deadline constraint to $D = 8$s and gradually increase the number of ingested instances $N$ from 32 to 30,000. As shown in Figure 7(b), the model equipped with the scheduler can adapt to the workload by switching between sub-models, which leads to higher effective accuracy. Specifically, when $W_{exp} = 100$, the serving model is the full model (i.e., the slice rate $r = 1$). When $W_{exp} = 10,000$, the serving model is switched to the smaller model of a slice rate 0.5 to avoid dropping instances. When $W_{exp} = 18,000$, where even the smallest sub-model of a slice rate 0.25 can not process all instances within the time limit. In such scenarios, the serving model is switched to the smallest sub-model to maximize the throughput. As shown in Figure 8(a), the system dynamically adapt the model size to increase the throughput until it reaches the maximum throughput, which is the same as the fastest sub-model.

We also measure the latency of the system, which is shown in Figure 8(b). Specifically, when the ingesting rate is low, since all sub-models now can meet the deadline, the scheduler will adopt the sub-model of slice-rate 1.0 for higher accuracy. When the ingesting rate reach around 10,000, both the sub-models of a slice-rate 1.0 and 0.75 cannot process all instances before the deadline. The combination of sub-models however, can meet the deadline.

<table>
<thead>
<tr>
<th>Ingesting rate (#instances/second)</th>
<th>Throughput (minibatches/second)</th>
<th>Tail latency (second)</th>
<th>Ingesting rate on scheduled models</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sub-model with $r = 1$</td>
<td>Sub-model with $r = 0.75$</td>
<td>Sub-model with $r = 0.5$</td>
<td>Combination of sub-models</td>
</tr>
<tr>
<td>Theoretical value</td>
<td>$\text{ave}_{r=0.25}$</td>
<td>$\text{ave}_{r=0.5}$</td>
<td>$\text{ave}_{r=1}$</td>
</tr>
</tbody>
</table>
constraint until the ingesting rate reach 18,000, where the serving model will entirely switch to the sub-model of a slice rate 0.25.

To better illustrates the combinations of the scheduled sub-models under different instance ingesting rates, we present detailed assignment of the mini-batches to the sub-models in Figure 8(c). We can observer that when the ingesting rate is low, the model assigns all mini-batches to the sub-model of a slice rate 1.0. Since the sub-model of a slice rate 0.75 and 0.5 have similar accuracy, while the sub-model of a slice rate 0.5 is much faster, the scheduling algorithm favors the later sub-model for achieving higher effective accuracy.

For the second scenario, where multiple models can be loaded to the system, SINGA-Easy can have multiple elastic models and can generate multiple combinations of sub-models. In contrast, Model-Switching can only have fixed combinations of models.

In conclusion, the experiments on effective accuracy, throughput, latency, and sub-model combinations confirm that the model trained with the model slicing technique and our proposed scheduling algorithm support dynamic workloads via finer-grained elastic computation control. It further illustrates the adaptability of SINGA-Easy.

5.3 Multimedia Applications
We further demonstrate the usability of SINGA-Easy on various applications. Due to the space limit, we showcase representative examples in Figure 9. The Singapore Food Detection component has been used to develop FoodLG app\(^6\), which is customized for healthcare applications such as pre-diabetes management and diet recommendation. For the training dataset, we crowdsource to knowledge users using CDAS[23] for labelling. Medical applications like X-ray-based diagnosis is shown in Figure 9(b), the GradCam map highlights the unhealthy areas with warm colors (red and purple). The LIME map circles the unhealthy areas with yellow color. The explanation maps can assist clinicians in verifying the correctness of the diagnosis, e.g., whether explanation maps match is in line with their diagnosis.

6 RELATED WORK
In this section, we review the related work of ML/DL systems and framework. Their are highly accessible and could be used to extend our SINGA-Easy.

PyTorch [28] can achieve automated ML using the Auto PyTorch library [45], but it does not provide the system infrastructure for ML life cycle management in multimedia applications. SINGA-Easy can be used to facilitate the PyTorch models.

Microsoft NNI\(^7\) is a ML framework supporting model compression. However, it does not provide elastic inference capabilities to the models. While the slice-rate in SINGA-Easy is more understandable.

Hopswork [17] is a data science platform for the design and operation of data analytics applications. The system applies HopsFS, a highly scalable distributed file system, to improve its efficiency. While our system focuses more on the usability to AI applications.

In summary, there are indeed many data analytics systems developed in recent years. However, their design philosophy and criteria can be very different. As introduced in Section 1, our SINGA-Easy is designed to improve the usability and adaptability in developing multimedia applications.

7 CONCLUSIONS
In this paper, we introduced SINGA-Easy - a learning system focusing on usability and adaptability. SINGA-Easy was built on top of Apache SINGA. It assists users in managing data and models, and developing AI applications. We have used SINGA-Easy to develop multi-media applications such as a chest X-ray image explanation function and food detection system. We showed that SINGA-Easy is highly extendable as it can be used with various third-party machine learning models.

Moving forward, we note that there exist other bottlenecks in data science such as data loading, visualization, cleaning, labeling, and data transformation. Future extensions to SINGA-Easy may include such data science supporting modules.

Acknowledgement: We thank the anonymous reviewers for their constructive comments and NUS colleagues for their comments and contributions. This research is supported by Singapore Ministry of Education Academic Research Fund Tier 3 under MOE’s official grant number MOE2017-T3-1-007. Meihui Zhang’s work is supported by the National Natural Science Foundation of China (62050099).

\(^6\)http://foodlg.com/

\(^7\)Microsoft NNI: https://github.com/microsoft/nni


