DOPE-ML: DESIGNING OPERATIONAL POLICIES FOR ML ENVIRONMENTS

Prasoon Patidar [1] **Sudershan Boovaraghavan** [1] **Balamurugan Marimuthu** [1]

ABSTRACT

The field of Machine Learning Operations (MLOps) has grown with time due to increasing number of ML application deployed worldwide. Over the years, people have developed many MLOps platforms(both open and closed source) which can manage entire lifecycle of an ML based application. These systems can utilize heterogeneity in specialized accelerators such as GPUs, TPUs, FPGAs, and custom ASICs to train and serve ML models. These accelerators exhibit heterogeneous performance behavior variety of model architectures. Being an emerging field, MLOps is rapidly gaining momentum amongst Data Scientists, ML Engineers and AI enthusiasts. However, the burden of choosing right combination of ML task and deployment backend to optimize efficiency, cost and energy falls on users of these systems, which is a big challenge for target users. In this work, we present DOPE-ML, a learning based framework which automates choice of deployment backends for wide variety of ML tasks, while taking into consideration user priorities. As an end goal, we wish to remove burden of developers using any MLOPs systems to choose deployment backends across wide variety of tasks.

1 INTRODUCTION

Machine learning operations (MLOps) are DevOps for machine learning processes. MLOps enables data scientists to collaborate and increase the pace of delivery and quality of model development through monitoring, validation, and governance of machine learning models. This is equivalent to how DevOps helps software engineers develop, test, and deploy software quicker and with fewer defects. MLOps supports the data science life cycle just as DevOps supports the application development life cycle. As such, MLOps is based on the principles of DevOps. Some examples of such systems are MLflow, Seldon-Core, Bento-ML, KubeFlow, MLRun, ZenFlow, etc. These platforms provide API for integrating a variety of modeling frameworks like PyTorch, TensorFlow, Java, Sklearn, Keras, SHAP, Fastai etc, and provide a unified approach to manage these modeling frameworks. Further, these MLOps frameworks can also provide the ability to deploy these models on a wide variety of backend and device infrastructures including VMs, Dockers, CPUs, GPUs, eGPUs, Cloud services, and FAAS instances, thus providing a wide variety of options. These deployment backends exhibit heterogeneous performance behavior in terms of cost, energy usage, latency, etc. across various modeling frameworks (see Figure 1). Furthermore, these factors might be dynamic in nature as the amount and type of workload on a particular resource change over time.

As the number of modeling frameworks and deployment backends increases, it might be difficult to deploy a variety of ML-based tasks across heterogenous hardware which can leverage systemic advantages of deployment backends to optimize for performance, cost, and energy. Further, different kinds of (ML-based) applications might have different kinds of requirements in terms of when requests are processed(periodic/ event-driven/bursty), what is the primary metric of concern for the task(i.e performance for security-related tasks, energy for maintenance-related tasks, etc.). Currently, there are no frameworks that provide a way to optimize deployment strategies across heterogeneous backends considering job-specific priorities.

In our work, we propose DOPE-ML, which acts as a wrapper around an underlying MLOps system and decides operational strategies(including deployment, runtime, etc) based on available resources, and task level policies. We develop an online learning framework that incorporates task capabilities(i.e it can run on GPU/FaaS/Docker etc), task priorities(Energy/Cost/Latency), and current resource utilization to decide deployment strategies for all incoming tasks.

2 RELATED WORK

A lot of recent work is focused on optimizing Machine Learning workloads in distributed clusters. Themis (Mahajan et al., 2020), and Tiresias (Gu et al., 2019) provide algorithmic approaches to optimize resource sharing for deep learning training in GPU clusters, whereas Gavel (Narayanan et al., 2020) provides a cluster level, policy-based mechanism to schedule tasks on heterogeneous clusters. Allox (Le et al., 2020a) utilizes task-level capabilities to provide best-effort scheduling for training across CPUs and GPUs, and SLAQ (Le et al., 2020b) explores quality-runtime tradeoffs across multiple training jobs to maximize

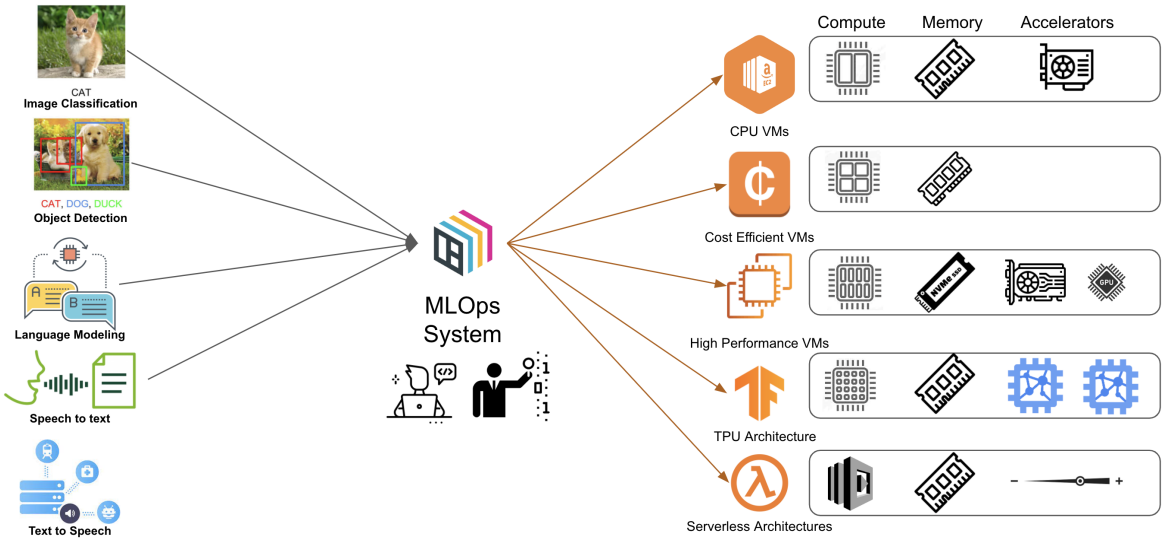

Figure 1. An Overview of the utility of MLOps systems. It allows deployment across a wide variety of applications, which utilize diverse ML models to cater to user requests without developing a separate end-to-end pipeline. In the current state-of-the-art systems, decisions for selecting appropriate deployment backend to fulfill incoming requests are manually taken by application and model developers, thus missing opportunities for fine-grained optimizations in terms of inference latency, the cost for users, and energy efficiency.

system-wide quality improvement. All these works focus on improving training efficiency and performance of deep learning models across high-performance distributed platforms. There are other works (Yang et al., 2018) that focus on the problem of resource allocation on scheduling in general learning-based workloads. Dynamo-ML (Chiang & Chou, 2021) is the closest work to what we are trying to achieve in our project. It proposes a set of runtime dynamic management techniques to handle the mixture of ML workloads to improve Kubernetes (an MLOps) systems performance across AWS GPU Clusters. However, it doesn't incorporate richness in deployment backends enabled by current MLOps systems and solely focuses on performance, i.e. not considering task level prioritization across a variety of metrics like cost, energy, and accuracy.

3 Our Contributions

We build on the top of BentoML, a well-known MLOps system. It is implemented in python and supports a very wide variety of serving backends. It is primarily used to simplify model deployment and enables serving models in production in minutes. It provides a unified mechanism to train ML models in a custom manner and provides framework-level API to package models in a unified format (called bentos), which are then manually deployed to different kinds of backends as needed. It also supports custom cloud backends like AWS Lambda, AWS Sagemaker, and Azure functions, which is not in the scope of this project. For project purposes, we focused more on hardware heterogeneity in terms of CPU usage, GPU availability and usage, and memory restrictions.

In our end-to-end system, We obtain the deployment motivation/requirement from the user as a high-level performance policy and not as a resource requirement policy. Each deployment is tagged with a user-provided SLA/Latency, accuracy, and energy/power requirements. Our implementation performs an automated initial offline profiling of the given model and learns the knobs to tune to meet the user's performance policy. Based on the model signature, we profile the model's behavior under varying workloads, and different hardware resources using a custom offline profiler. Additionally, we also profile the model's characteristics based on its compute, network, and memory requirement properties. Combining both the impact data and the characteristics data, we utilize an online learning approach to allocate and assign just the right resources that will satisfy the user's performance policy, maximizing resource utilization and optimizing for future requests.

In order to avoid Dope-ML model training from scratch, we utilize public trace datasets from big cloud providers like Alibaba (ali) to pre-train our model. Unfortunately, these datasets do not provide sufficient information about the type of user requests and virtually no information about user priorities. We overcome this challenge by utilizing model performance datasets like MLPerf (mlp), which provide benchmarking information of popular open-sourced machine learning models. More details are present in Section 4.

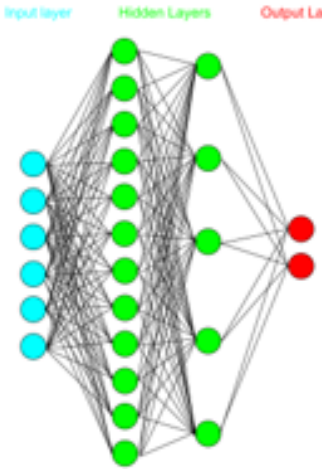

Figure 2. MLP Model

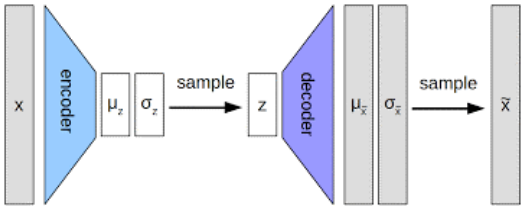

Figure 3. VAE Model

The effectiveness of our methodology is evaluated by comparing a set of SLA, resource utilization, and power metrics of SOTA/widely used deployment mechanisms and our mechanism for a given model (e.g MLPerf) and its performance requirement under different workloads, available hardware resources, and co-located jobs.

4 Modeling Approach

Dope-ML takes a machine learning approach to predict the resources needed for the given application and model. The basic requirement of the model is to take in the user requirements and the model characteristics as input and predict the optimal resource requirements viz. CPU & GPU utilization and memory usage respectively as output. We take a couple of different approaches to meet the requirement. The first approach is a simple vanilla regression model whereas the second one is a Variational Auto Encoder (VAE) model. The VAE model describes input features into a latent space using which we can conditionally predict resources. Both the models are similar to multi-output regression with 4 dependent output values which translate to 4 independent training loss values during training.

4.1 Existing Datasets

Today there are several tech companies that are deploying large ML-as-a-Service (MLaaS) clouds, often with heterogeneous GPUs, to provision a host of ML applications. To train and test the proposed model, we need adequate real-world data which is representative of the Machine Learning workloads in the real-world MLOps systems. Thus, obtaining such real-world ML workloads that run on heterogeneous machine configurations raises a number of challenges.

- *Lack of Real-World Traces for ML Workloads:* Obtaining a real-world ML Workload trace consisting of training and inference jobs collected from large production machines is not easily available and is not available.
- *Application-specific ML Workload Scheduling:* There is a lack of ML workload traces that capture resource characteristics allocated to different applications. For example, several applications such as object detection, etc, predominantly use different Applications and have different workload characteristics.
- *Diversity in ML Model Workloads:* There is also a need for traces that capture diversity in resource characteristics for the same workload trace.
- *Diversity in Input for different ML Workloads:* In addition, the variation in the size of the input for the same Machine Learning model results in varying resource consumption.

Thus, to overcome these challenges: we explored both real-world and synthetics datasets for real-world profiling.

4.1.1 Real-World Datasets

We explored several datasets that include real-world traces of information ML workloads from Microsoft Azure(Cortez et al., 2017; Hadary et al., 2020) and Google Borg (Verma et al., 2015). However, several of these traces consisted of only the virtual machine (VM) workloads and their resource usage data for a variety of jobs that ran in those clusters. The Alibaba cluster trace dataset that is publicly available is the one closest to our needs. The released trace contains a hybrid of training and inference jobs running state-of-the-art ML algorithms. It is collected from a large production cluster with over 6,500 GPUs (on 1800 machines) in Alibaba PAI (Platform for Artificial Intelligence) (Weng et al., 2022), spanning the July and August of 2020.

Alibaba PAI Trace dataset overview:

The released PAI trace (Weng et al., 2022) contains the arrival time, completion time, resource requests, and usages on GPUs, CPUs, GPU memory, and main memory of the

task_name	workload	runtime_i	cpu_usage	gpu_wrk_util	max_mem	gpu_type
tensorflow	bert	305.0	27.711864	23.758065	3.552734	MISC
PyTorchWorker	bert	12315.0	107.478279	169.969181	119.771484	V100M32
worker	bert	78.8	244.873074	1.441348	12.208984	T4
worker	bert	7615.0	4307.133290	0.000000	11.160156	T4
PyTorchWorker	nmt	12126.0	988.092784	34.602965	14.343750	P100

Figure 4. Overview of the Alibaba Dataset

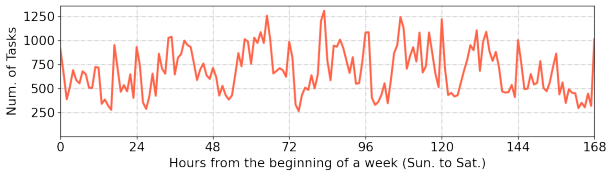

Figure 5. Characteristics of the Alibaba dataset: Number of tasks submitted and their instances in one week

workloads at various levels (e.g., job, task, and instance). Table 4 shows an example of traces from the Alibaba dataset. The trace records the arrival time, completion time, resource requests, and usages in GPUs, CPUs, GPU memory, and main memory of the workloads at various levels (e.g., job, task, and instance).

Figures 5 and 6 show the tasks and instance submissions, as well as the overall resource requests in one week during the trace collection period. We see in Figure 5 that in addition to daytime, midnight is also a rush hour for task submissions. However, in Figure 6 we see that tasks submitted at midnight are less compute-intensive, having only a few instances and requesting a small amount of resources. The Alibaba real-world trace of ML workload not only provided information about different resource usages, but it also provided us with an interesting insight to develop our model. However, one of the main challenges of this dataset was the lack of availability of information about the type of application workloads. Although the data set provided information about the model and the GPU on which the particular trace was run, it did not provide any information about which type of application was using the workload.

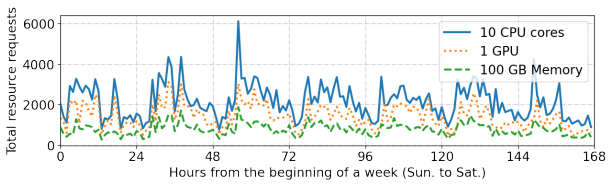

Figure 6. Characteristics of the Alibaba dataset: Total resource requests of running tasks in one week.

Task	Reference Model	Data Set	Quality Target
Image classification (heavy)	ResNet-50 v1.5 25.6M parameters 7.8 GOPS / input	ImageNet (224x224)	99% of FP32 (76.456%) Top-1 accuracy
Image classification (light)	MobileNet-v1 224 4.2M parameters 1.138 GOPS / input	ImageNet (224x224)	98% of FP32 (71.676%) Top-1 accuracy
Object detection (heavy)	SSD-ResNet34 36.3M parameters 433 GOPS / input	COCO (1,200x1,200)	99% of FP32 (0.20 mAP)
Object detection (light)	SSD-MobileNet-v1 6.91M parameters 2.47 GOPS / input	COCO (300x300)	99% of FP32 (0.22 mAP)
Machine translation	GNMT 210M parameters	WMT16 EN-DE	99% of FP32 (23.9 SacreBleu)

Figure 7. Overview of MLPerf benchmark Application Tasks

4.1.2 *Using MLPerf to model Alibaba Workload Traces to Application*

Thus, to overcome this, we used MLPerf benchmark metrics to model the Alibaba workload traces to a certain application task such as Image Classification, Object Detection, etc. MLPerf (Mattson et al., 2020) is a full system benchmark, testing machine learning models, software, and hardware with broad industry and academic support.

Figure 7 shows the applications of ML models that MLPerf has benchmarked. We use these MLPerf benchmark results to model the input data for different application tasks. Thus, based on our explorations, our input features and output values using the cleaned Alibaba dataset are as follows:

In features: 'task_name', 'workload', 'no_of_inferences', 'plan_cpu', 'plan_gpu', 'plan_mem'

Predicted outputs: 'cpu_usage', 'gpu_wrk_util', 'max_mem', 'max_gpu_wrk_mem', 'gpu_type'

4.1.3 *Synthetic Dataset from Profiling*

However, as mentioned earlier, it doesn't provide any information on the input workload i.e number of inputs, input size, model dimensions, etc. This makes it difficult to use this model to fit our needs. To try using this model we assumed the 'plan_cpu', 'plan_gpu' to be representative of the workload characteristics.

To overcome the drawbacks of the publicly available datasets, we proceed to create our own synthetic dataset with additional necessary features as mentioned earlier. The workflow for profiling and obtaining the synthetic dataset is shown in figure 7.

We deploy a given model across several combinations of inputs and deployment backends and obtain the run characteristics. This allows us to have the deployment model characteristics and input characteristics as input features for our prediction model.

In features: 'app_name', 'model_name', 'model_type',

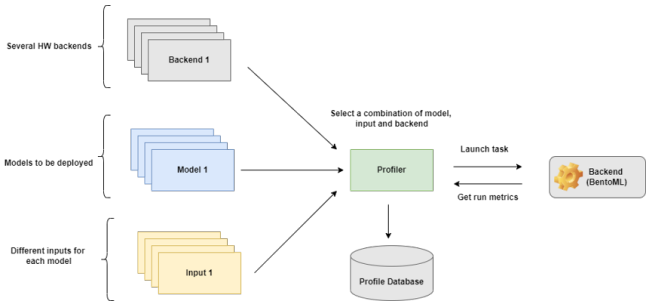

Figure 8. Dataset obtained from Profiling

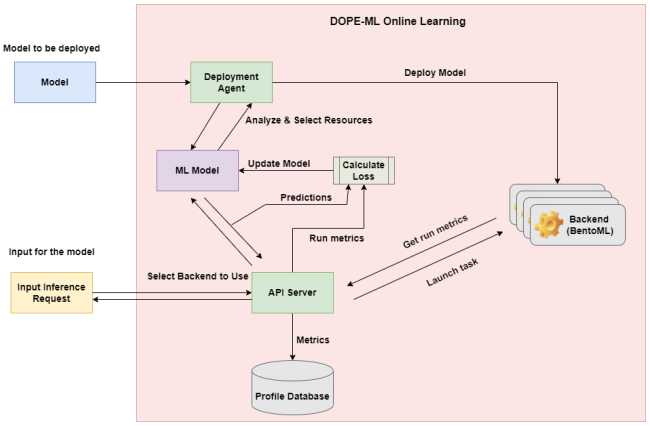

Figure 9. Online Learning Flow

'model_layers', 'model_hiddens', 'inf_time', 'input_dimens'

Predicted outputs: 'cpu_util' , 'gpu_util', 'cpu_mem' , 'gpu_mem'

4.2 Online Learning

We extend the pretraining approach with an online-learning mechanism where the resource prediction model gets updated during its production usage as described in figure 8. The prediction loss is calculated using the actual run characteristics and the model is updated on the fly. This allows the model to grow as the type and nature of workload get varied over time.

5 Implementation

5.1 Remote Backend Deployment

In this section, we discuss our end-to-end implementation of Dope-ML on top of the MLOps system. In section 5.2, We talk about user applications we selected along with Machine learning models to simulate real-life user requests, and in section 5.1, we discuss our deployment strategy for these user applications with Bento-ML along with Dope-ML integration.

5.2 User Applications

To simulate real-life workload in online learning, we created 5 applications that simulate most frequently used machine learning tasks in day-to-day life.

1. **Image Classification**: It is one of the most utilized, and studied machine learning tasks in the last decade. Image classification models are used in many areas, such as medical imaging, object identification in satellite images, traffic control systems, brake light detection, machine vision, and more.

2. **Object Detection**: Object tracking has a variety of uses, some of which are surveillance and security, traffic monitoring, video communication, robot vision, and animation. Face detection and Face Recognition is widely used in computer vision task.

3. **Speech to Text**: Speech recognition technologies such as Alexa, Cortana, Google Assistant, and Siri are changing the way people interact with their devices, homes, cars, and jobs. The technology allows us to talk to a computer or device that interprets what we're saying in order to respond to our question or command. Speech recognition technology and the use of digital assistants have moved quickly from our cellphones to our homes, and its application in industries such as business, banking, marketing, and healthcare is quickly becoming apparent.

4. **Text to Speech**: Text translation allows users to see text and hear it read aloud simultaneously. There are many apps available, but typically as text appears on the screen, it's spoken. Some software uses a computer-generated voice and others use a recorded human voice. Very often the user has a choice of gender and accent as well. To convert text to speech, we are using technotron models, which are further decoded into the audio signals using wave RNN models.

5. **Language Modeling**: When deep learning is combined with NLP, a variety of interesting applications get developed. Language translation, sentiment analysis, name generation, etc., are some of these interesting applications. one of such interesting applications is masked language modeling. Masked image modeling is a way to perform word prediction that was originally hidden intentionally in a sentence. Masked language modeling and image modeling can be considered similar to autoencoding modeling which works based on constructing outcomes from unarranged or corrupted input. These models are used where we are required to

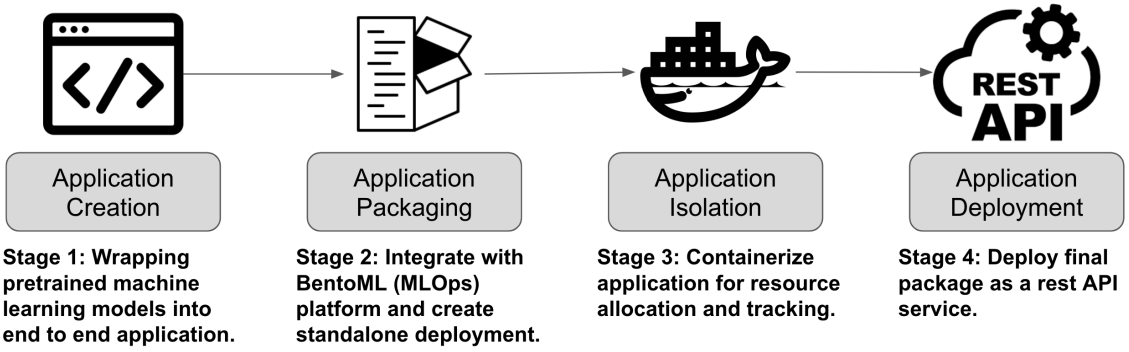

Figure 10. Steps involved in end-to-end deployment for all applications.

predict the context of words. Since the words can have different meanings in different places the model needs to learn deep and multiple representations of words. These models have shown improved performance levels in the downstream tasks such as syntactic tasks that require lower layer representation of certain models in place of a higher layer representation.

We have implemented these models as standalone applications and deployed them using BentoML in a remote backend to be served as a REST API requests. Details of remote deployment at an application level are provided in the following section.

Dope-ML is built on the top of BentoML. It is implemented in python and supports a very wide variety of serving backends. It is primarily used to simplify model deployment and enables serving models in production in minutes. It provides a unified mechanism to train ML models in a custom manner and provides framework-level API to package models in a singular deployment format, called Bentos. Following is the four-step process we followed to deploy our application as a rest API.

- **Stage 1(Creating Applications)**: We start by finding appropriate pre-trained models for all applications. For image classification, we used resnet () as a representative application for our implementation. For object detection, we used Single Shot Detection (SSD) model for our implementation due to its efficient execution. For speech recognition, we utilized pre-trained Wav2Vec models from torch audio libraries for simulating speech recognition models. To convert text to speech, we are using technotron models, which are further decoded into the audio signals using a wave RNN models. We have implemented Masked language modeling using pre-trained Bert model from google.
- **Stage 2(Convert Applications to Services(Bentos))**: We start by converting all pre-trained models used in an application into BentoML models. Further, these BentoML Models are embedded into services with pre-processing steps which convert raw user input(images/audio signals/ text inputs, etc.), and post processing steps that involves converting raw prediction into human interpretable predictions.
- **Stage 3(Containerize Applications for resource tracking)**: The amount of resources a service can utilize is important for optimizing backend selection for incoming requests. One way to do this is to initialize each application and type of backend based on CPU/GPU utilization into separate virtual machines. This explodes quickly as the number of applications or potential backends grows. Thus, we have containerized all applications with dockers, which can be then initialized with resource constraints on CPU, GPU, and memory.
- **Stage 4(Applications Deployment)**: In our implementation, we emulated heterogeneity in hardware using a variety of CPU, GPU, and memory constraints. Table 1 show 4 sets of backends we utilize to deploy all application. We create these 4 backends with the rationale of emulating high constraints and low constraint settings and the impact of hardware accelerators on user request processing. All the deployments are done on one large VM with 64 Cores, 64GBs of RAM, and 2 NVIDIA 1080 Ti GPUs. Applications are containerized and assigned to a specific CPU index to remove any task co-location possibilities. Currently, utilizing different types of architectures is out of project scope as bentoML does not support custom cloud environment deployments currently.

Backends	Num CPUs	Memory(in GBs	Num GPUs
B1: Low Resources(w/o GPU)	2	2-4 GBs	0
B2: High Resources(w/o GPU)	4	4-6 GBs	0
B3: Low Resources(w/ GPU)	2	2-4 GBs	1 (11GBs GPU Memory)
B4: High Resources(w/ GPU)	4	4-6 GBs	1 (11GBs GPU Memory)

Table 1. The diverse set of backends deployed for applications in our implementation shows a) differentiation between constrained and unconstrained environments and b) the impact of hardware accelerators like GPUs.

6 EVALUATION

6.1 Pretraining Evaluation

6.1.1 Using Modified Alibaba Dataset

We initially trained and evaluated our MLP model using the modified Alibaba dataset. As mentioned in section 4.1.1, key features such as model details and workload details (input size/batch size) were not available in the dataset. The result obtained is shown in figure 10. Scaled Mean Squared Error (MSE) was used as the loss function and Mean Absolute Percentage Error (MAPE) was used as a measure of the regression accuracy or closeness of the prediction. With Alibaba data, the MAPE numbers seem to be on the higher side indicating lower accuracy. We believe this stems from the inadequate input features in the Alibaba dataset.

6.1.2 Using the Synthetic Dataset

The proposed MLP prediction model was trained using the synthetic dataset that is obtained as mentioned in section 4.1.3. Figure 11 shows separate CPU utilization, and memory usage prediction results. The results obtained with the synthetic dataset are far better than the Alibaba dataset. This proves that with more input features, the model is able to predict well. The MAPE value between the Synthetic dataset and Alibaba dataset shows a 9x improvement in accuracy.

6.1.3 Challenges and Learnings

Loss Calculation

This resource prediction problem is similar to a multi-output regression problem and comes with all the challenges associated with it. Unlike a simple single value regression, the different outputs predicted here are of different units and scales but are interrelated. We calculate losses for each of the different variables predicted and sum them up as the final loss value for training. We tried several loss calculation approaches to get reasonable predictions with the MLP model, which was quite challenging. Based on our trials, we observed that scaled (normalized) MSE and MAPE help the model learn better in this scenario. We plan to use this learning to further explore different model architectures that will increase the accuracy of the resource predictions.

Representative Data

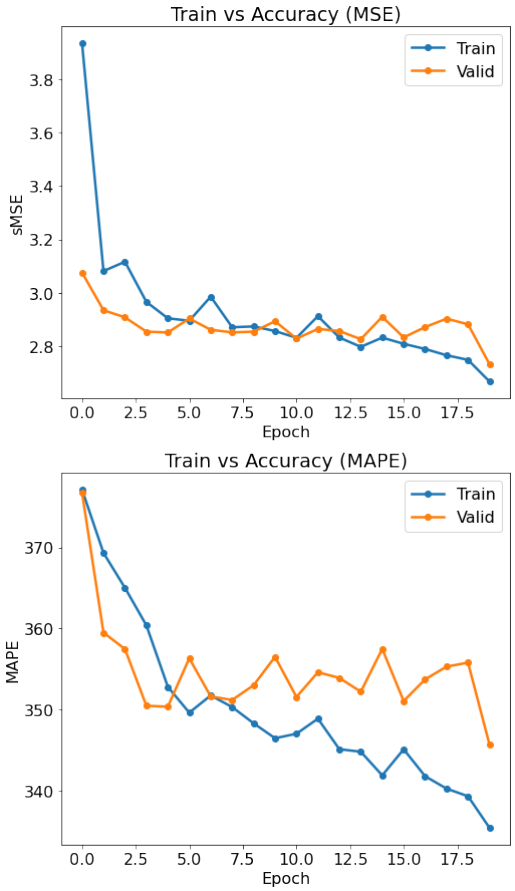

Figure 11. Evaluation of the MLP model trained using the Alibaba Dataset

Obtaining reliable and adequate data for fine-grained resource prediction is still a challenging problem. One way to tackle this is to take an online-learning approach. This not only provides diverse data but also makes it more realistic as it is directly from a production workload. We were able to implement and test the working of the online learning workflow proposed in section 4.2. Extensive evaluation of the online learning flow and evaluation of the eventual cost and resource utilization savings are planned to be done as future work.

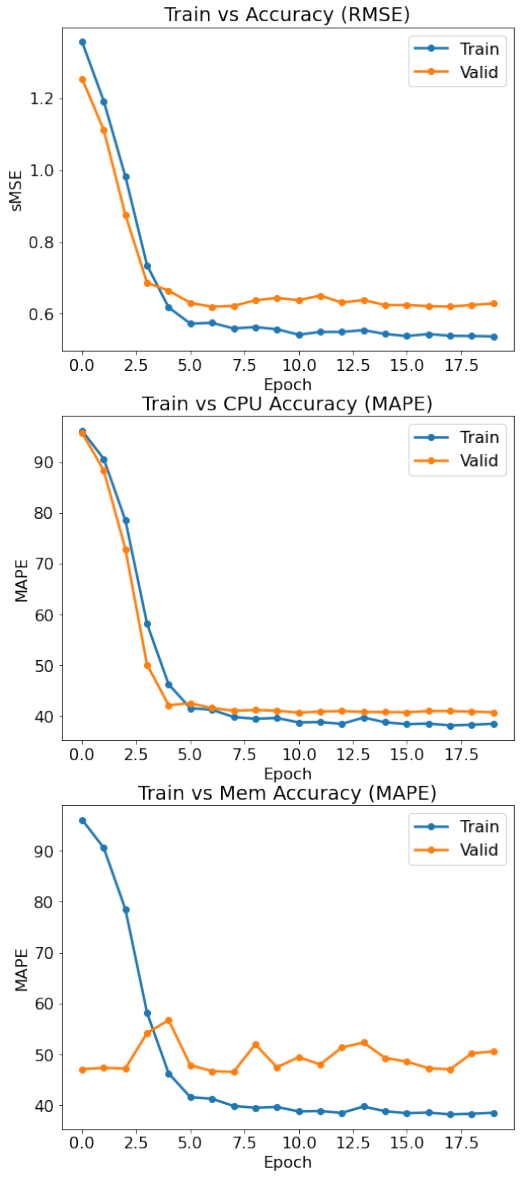

Figure 12. Evaluation of the MLP model trained using the Synthetic Dataset

7 Conclusion

In this paper, we proposed Dope-ML, a wrapper around an emerging MLOps system that can reduce the burden of selecting an appropriate backend to fulfill user requests for ML-based workloads. We designed an online learning approach to predict backend requirements and implemented our solution with BentoML and common representative applications. Our evaluation shows that our approach could learn request patterns and resource requirements and provide the best backend at the user request level.

References

Alibaba trace dataset for ml workloads. https://github.com/alibaba/clusterdata.

Mlcommons: Machine learning innovation to benefit everyone. https://mlcommons.org/en/.

Chiang, M.-C. and Chou, J. Dynamoml: Dynamic resource management operators for machine learning workloads. In *CLOSER*, pp. 122–132, 2021.

Cortez, E., Bonde, A., Muzio, A., Russinovich, M., Fontoura, M., and Bianchini, R. Resource central: Understanding and predicting workloads for improved resource management in large cloud platforms. In *Proceedings of the 26th Symposium on Operating Systems Principles*, pp. 153–167, 2017.

Gu, J., Chowdhury, M., Shin, K. G., Zhu, Y., Jeon, M., Qian, J., Liu, H., and Guo, C. Tiresias: A GPU cluster manager for distributed deep learning. In *16th USENIX Symposium on Networked Systems Design and Implementation (NSDI 19)*, pp. 485–500, Boston, MA, February 2019. USENIX Association. ISBN 978-1-931971-49-2. URL https://www.usenix.org/conference/nsdi19/presentation/gu.

Hadary, O., Marshall, L., Menache, I., Pan, A., Greeff, E. E., Dion, D., Dorminey, S., Joshi, S., Chen, Y., Russinovich, M., and Moscibroda, T. Protean: VM allocation service at scale. In *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*, pp. 845–861. USENIX Association, November 2020. ISBN 978-1-939133-19-9. URL https://www.usenix.org/conference/osdi20/presentation/hadary.

Le, T. N., Sun, X., Chowdhury, M., and Liu, Z. Allox: Compute allocation in hybrid clusters. In *Proceedings of the Fifteenth European Conference on Computer Systems*, EuroSys '20, New York, NY, USA, 2020a. Association for Computing Machinery. ISBN 9781450368827. doi: 10.1145/3342195.3387547. URL https://doi.org/10.1145/3342195.3387547.

Le, T. N., Sun, X., Chowdhury, M., and Liu, Z. Allox: Compute allocation in hybrid clusters. In *Proceedings of the Fifteenth European Conference on Computer Systems*, EuroSys '20, New York, NY, USA, 2020b. Association for Computing Machinery. ISBN 9781450368827. doi: 10.1145/3342195.3387547. URL https://doi.org/10.1145/3342195.3387547.

Mahajan, K., Balasubramanian, A., Singhvi, A., Venkataraman, S., Akella, A., Phanishayee, A., and Chawla, S. Themis: Fair and efficient GPU cluster scheduling. In *17th USENIX Symposium on Networked Systems Design and Implementation (NSDI*

20), pp. 289–304, Santa Clara, CA, February 2020. USENIX Association. ISBN 978-1-939133-13-7. URL https://www.usenix.org/conference/nsdi20/presentation/mahajan.

Mattson, P., Reddi, V. J., Cheng, C., Coleman, C., Diamos, G., Kanter, D., Micikevicius, P., Patterson, D., Schmuelling, G., Tang, H., et al. Mlperf: An industry standard benchmark suite for machine learning performance. *IEEE Micro*, 40(2):8–16, 2020.

Narayanan, D., Santhanam, K., Kazhamiaka, F., Phanishayee, A., and Zaharia, M. Heterogeneity-Aware cluster scheduling policies for deep learning workloads. In *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*, pp. 481–498. USENIX Association, November 2020. ISBN 978-1-939133-19-9. URL https://www.usenix.org/conference/osdi20/presentation/narayanan-deepak.

Verma, A., Pedrosa, L., Korupolu, M., Oppenheimer, D., Tune, E., and Wilkes, J. Large-scale cluster management at google with borg. In *Proceedings of the Tenth European Conference on Computer Systems*, pp. 1–17, 2015.

Weng, Q., Xiao, W., Yu, Y., Wang, W., Wang, C., He, J., Li, Y., Zhang, L., Lin, W., and Ding, Y. {MLaaS} in the wild: Workload analysis and scheduling in {Large-Scale} heterogeneous {GPU} clusters. In *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*, pp. 945–960, 2022.

Yang, R., Ouyang, X., Chen, Y., Townend, P., and Xu, J. Intelligent resource scheduling at scale: a machine learning perspective. In *2018 IEEE symposium on service-oriented system engineering (SOSE)*, pp. 132–141. IEEE, 2018.