COMS 6156 - Topics in Software Engineering
Spring 2024 - Final Report
Mike Segal | ms6135

Enhancing MLOps with Kubeflow, GitHub Actions, AutoML, and SHAP

Synopsis

MLOps, which we can define as the management and automation of the entire Machine Learning (ML) lifecycle, is becoming a more mature discipline, with major cloud providers such as Google and Amazon creating offerings (Vertex AI and Sagemaker respectively) that enable engineers to automate their machine learning pipelines. The purpose of this project was to evaluate the capability of an open-source MLOps platform, Kubeflow, to successfully construct an ML pipeline, and to determine whether AutoML and model interpretability packages could be used to improve model performance. By using a grid algorithm for hyperparameter tuning, we demonstrate that AutoML is a viable tool for improving ML model performance without requiring direct human supervision or activation. Furthermore, by utilizing SHAP, a model interpretability package, we provide an example of how model evaluation and validation can be improved. Finally, to ensure full automation, we demonstrate how GitHub Actions can be leveraged to automate pipeline runs and create an MLOps lifecycle that requires minimal human supervision to ingest data, conduct pre-processing, train and tune the model, and validate that the model is performing correctly using interpretability.

To evaluate the applicability of Kubeflow to evaluate different modalities, we trained Natural Language Processing (NLP) and Computer Vision (CV) models. Specifically, we trained a BERT model on the SQuAD dataset for the NLP component, and trained a Resnet-18 model for the ImageNet dataset. To ensure that our code was correct, we first trained the models using a traditional machine learning approach, without utilizing the Kubeflow component and pipeline functionality. To minimize variation between Kubeflow and non-Kubeflow environments, we conducted all of our training using the same Google Cloud Kubernetes cluster configurations, with the same CPU and GPU configurations. This required creating additional node pools within Kubernetes that were configured with GPU functionality, which necessitated resolving Google Cloud resource availability challenges, and reconfiguring the Kubeflow deployment.

Both models were trained in a similar way: the dataset was downloaded, data quality checks were conducted to ensure that no values were missing, the data was preprocessed, and the models were trained and validated. While we originally intended to utilize SodaCore for data quality checks, that would have required more extensive database integration that was beyond the scope of this project. Instead, we created custom functionality to check for null values and to ensure that the data was structured in the expected way, and applied these checks to both datasets.

Once the traditional models were trained and evaluated, we then restructured the code into Kubeflow components and pipelines, and configured the docker images for each of the components. Since Kubeflow runs each component as a dockerized container, the engineer has the ability to only use the configurations required for that part of the process, potentially optimizing overall system performance.

The pipelines were then compiled using Kubeflow, generating the YAML configurations, and starting the model training process. The Kubeflow and traditional training runs required approximately the same amount of time to train: the NLP model trained on the SQuAD dataset was successfully trained in eight hours, while the CV model trained on the ImageNet dataset was successfully trained in six hours. All training was performed using Google's N1-Standard-4 virtual machines, enabled with 4 T4 GPU's.

At this point, we began to differentiate the models to facilitate our ablation study and explore whether Katib AutoML would improve the model performance. The traditional and Kubeflow models were evaluated on their performance, which demonstrated values consistent with the existing literature. We then began to apply Katib's grid search algorithm for hyperparameter tuning to improve the performance of both the SQuAD and ImageNet models. While the models demonstrated a marginal performance improvement, it may not have justified the additional compute resources required for the process.

To improve the model interpretability, we also utilized SHAP as a validation step. While the outputs were not deterministic, they still allowed us to validate that the models were performing as expected. For the ImageNet model, SHAP generated the most likely labels for images, enabling the engineer to quickly assess if the model was performing adequately. In turn, using SHAP to interpret the SQuAD model showed the tokenization and predicted label that is expected in the NLP process.

As a final step in the process, we integrated GitHub Actions to conduct pipeline runs both as a scheduled cron job and on a GitHub commit. While the pipelines were not run in full, we validated that they were initialized correctly. The GitHub Actions integration demonstrated that the entire machine learning pipeline could be fully automated, such that in a real deployment, scheduled data ingestion could be used to trigger the retraining of a machine learning model, thereby enabling an enterprise to support optimal performance and minimize model drift.

The project can therefore act as a template for how to deploy full MLOps solutions within Kubeflow, and can assist engineering teams in automating their Machine Learning workflows. However, as we will discuss in later sections, the capabilities available in the organic cloud offerings, such as Google's Vertex AI and Amazon's Sagemaker, are significantly more advanced than Kubeflow, with much broader functionality and less limitations on workload management. While Kubeflow does allow MLOps to be successfully performed, it requires considerably more effort, such that any cost savings from using an open-source platform are rapidly eclipsed by unoptimized compute costs and lost productivity.

Research Questions

I. Can AutoML for hyperparameter tuning improve model performance when employed as part of an ML pipeline?

In order to determine if incorporating AutoML into MLOps would be a viable option for engineering teams, we created an ablation study that measures model performance before and after AutoML hyperparameter tuning using a Grid algorithm. This algorithm is fairly primitive, and involves providing a range of learning rate and momentum values, along with increments for the algorithm to systematically test, until the lowest loss is determined. While this approach may not be the most sophisticated, it also has a high likelihood of finding optimal hyperparameters for a model. To conduct the grid search on the SQuAD model, we had to train the model for each hyperparameter setting, such that for eight settings and an approximate training time of eight hours per run, the total time to fine-tune the model was approximately 65 hours. In turn, fine tuning the ImageNet model with the same grid algorithm required approximately 50 hours. Parallelization was not employed in order to gain a better understanding of the baseline, but future research in the area could leverage this approach. All eight trials and their outcomes are shown in figure 1. As we can see, the loss was minimized on the SQuAD model with a learning rate of 0.45 and a momentum of 0.7.

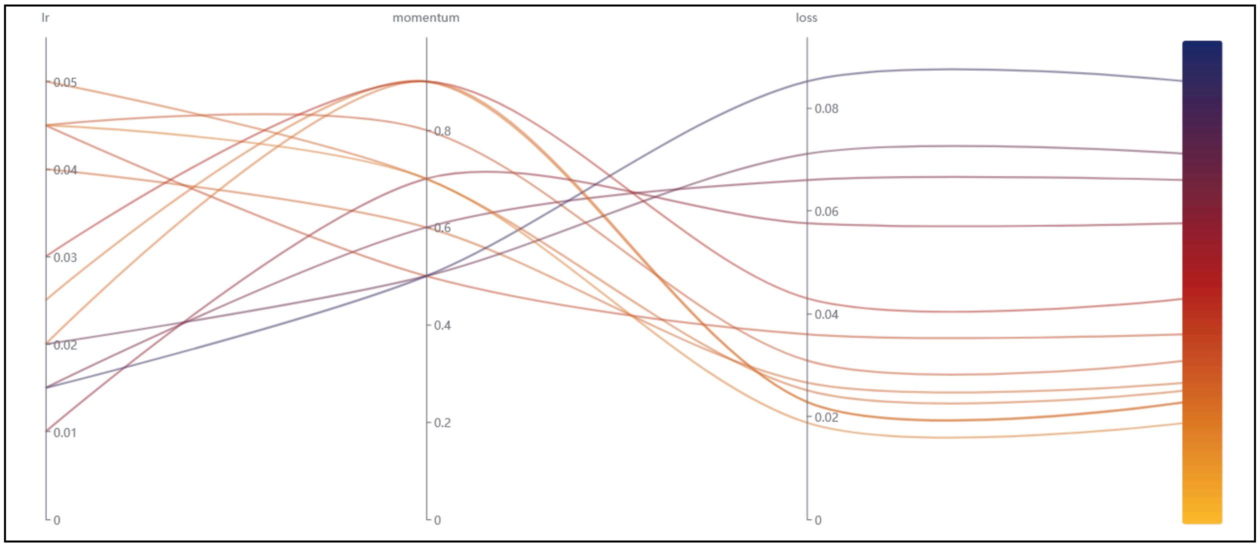


Fig. 1: AutoML running the Grid Algorithm for SQuAD Hyperparameter Optimization

Once the best performing model was identified, we proceeded to conduct statistical analysis of its performance, and compare it to the base model. There was some increase in the performance of both models, indicating that AutoML did succeed in improving the model performance.

However, from the enterprise perspective, especially with regards to cost optimizations, it is important to consider how expensive the hyperparameter tuning can be, and whether alternative strategies can be employed to improve model performance. In the case where models need to be highly precise, it is quite possible that fine tuning using AutoML is a viable and necessary part of the MLOps life cycle, but there are also cases where the engineering team should consider the costs associated with multiple retraining runs, and whether finding a more effective model architecture should take precedence.

As we can see in Table 1, the ImageNet base model achieved approximately the expected literature values on the Top-1 and Top-5 accuracy benchmarks, where Top-1 is the model outputting the top predicted label for an image, and Top-5 is the model outputting the correct label in the top 5 predicted labels. After running the AutoML hyperparameter tuning, the model performance slightly improved, and we can see that after conducting a t-test, the P-value was significantly lower than the 0.05 threshold for statistical significance on both the Top-1 and Top-5 accuracy tests. Consequently, we can see that AutoML did provide a statistically significant improvement of the ImageNet base model when trained using the ResNet-18 architecture.

ImageNet Metric	**Base Model**	**After AutoML**	**P-value**
Top-1 Accuracy	69.7 - 70.6 %	71.4 - 72.1 %	0.00000998
Top-5 Accuracy	89.0 - 89.3 %	89.5 - 89.7 %	0.00088

Table 1: ImageNet Model Performance Statistics

Using AutoML to tune the hyperparameters on the SQuAD-BERT model had a similarly statistically significant effect on the model's accuracy, precision, recall, and F1 score. The base model's performance were all within expected ranges against the literature values, and showed slight improvement after AutoML, as we can see in Table 2.

SQuAD Metric	Base Model	After AutoML	P-value
Accuracy	80.8 - 81.4 %	81.8 - 82.1 %	1.14e-28
Precision	88.1 - 88.3 %	88.5 - 88.7 %	4.86e-32
Recall	88.6-88.9%	88.7-89.1%	1.06e-12
F1 Score	87.8-88.5%	87.9-88.5%	2.25e-02

Table 2: SQuAD Model Performance Statistics

It should be noted, however, that the latest models trained on the ImageNet dataset (such as OmniVec) are achieving Top-1 accuracy rates of 91.4%, which is far beyond what the ResNet-18 architecture accomplished. Similarly, the IE-NET ensemble model achieved an F1 score of 93.214 in 2021, which is significantly higher than the SQuAD-BERT architecture we employed. Therefore, in the case of an engineering team considering how best to optimize their model's performance, considering alternative architectures should be the first choice, on the condition that the other architectures are computationally feasible for the organization. Only once that is evaluated should the team proceed with hyperparameter tuning. For the purpose of answering our research question, we can see that utilizing AutoML as part of an MLOps pipeline can in fact result in statistically significant improvements in model performance, provided that the engineering team is willing to devote the computational resources required to do so.

II. Can model interpretability using SHAP be employed as a validation step prior to model deployment?

As machine learning models are becoming increasingly ubiquitous, understanding how they reason and reach their conclusions is becoming increasingly vital. To offset the historically black box nature of machine learning models, we wanted to evaluate whether model interpretability libraries can provide the needed clarity for engineering teams. Furthermore, we wanted to determine if deploying a model interpretability module would be a viable option as part of an MLOps pipeline.

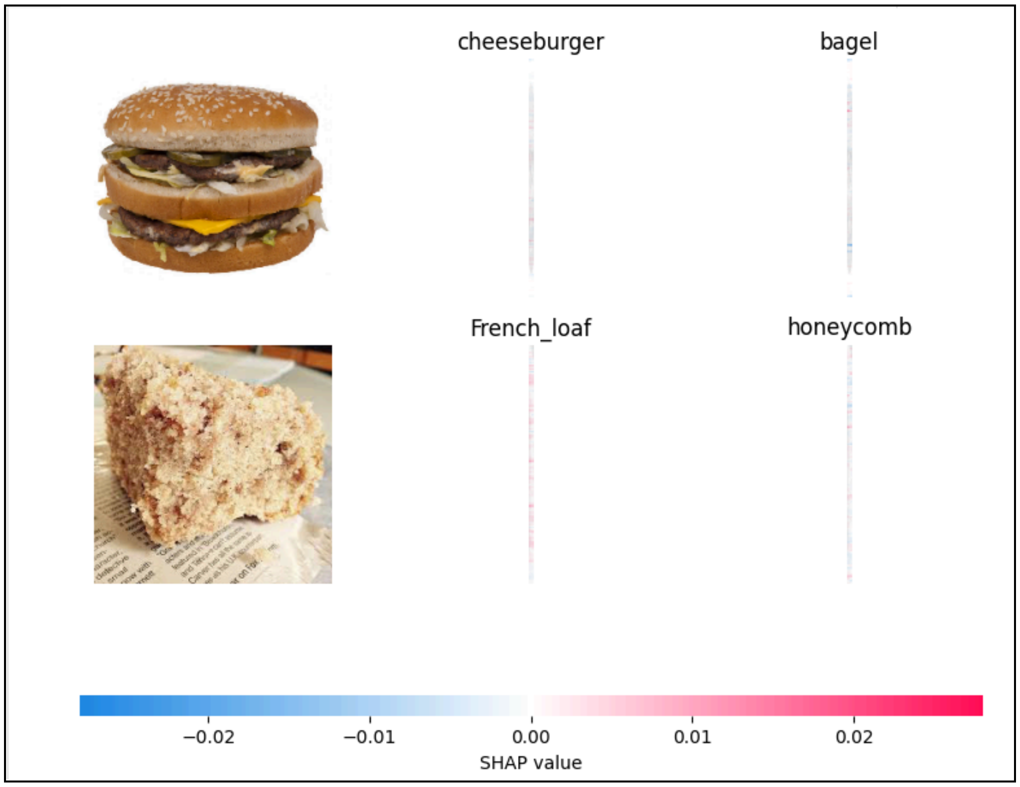
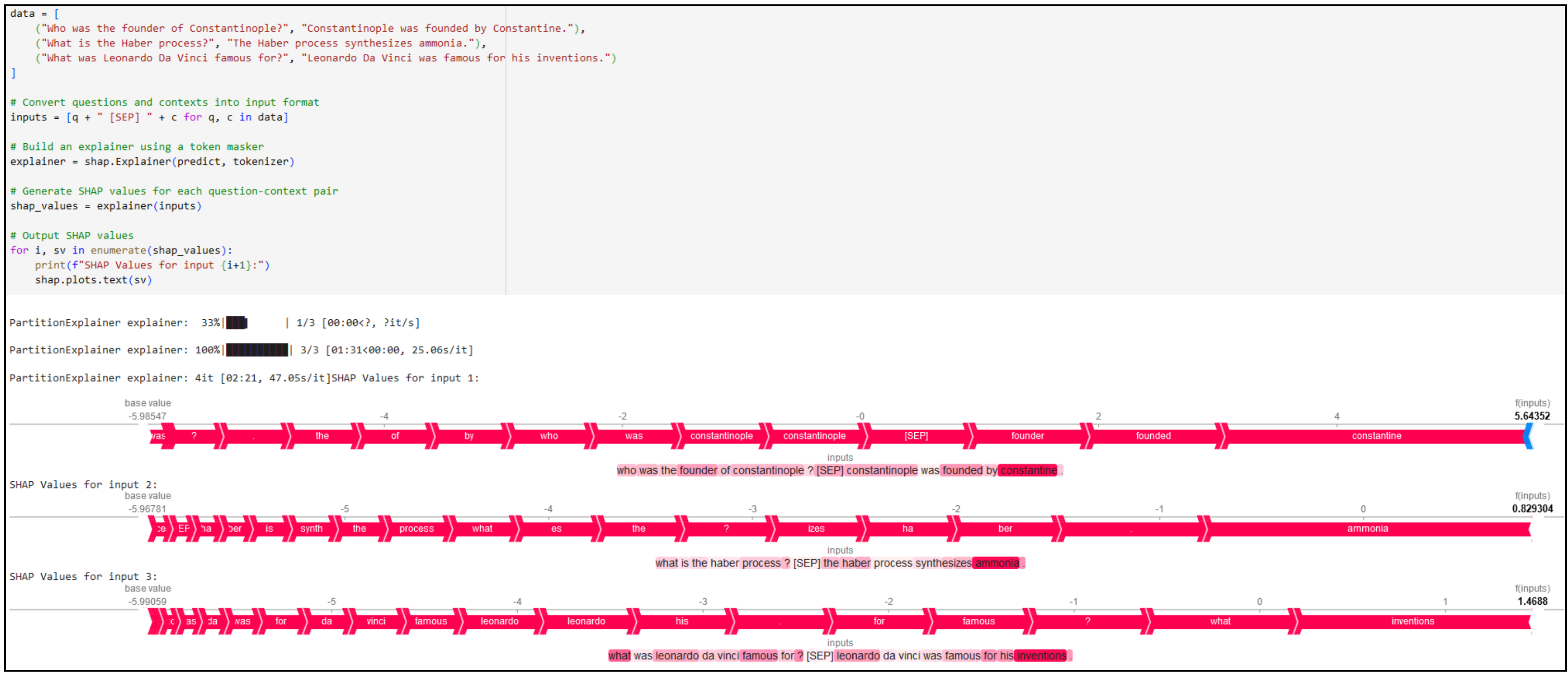


Fig. 3: ImageNet Model Interpretability Using SHAP

As we can see in figure 3, the ImageNet model was easily interpreted using the SHAP package, and we can see that the Top-1 prediction for both images appears accurate. While the computer vision functionality doesn't provide as robust of an explanation, we can expect the functionality to improve over time. Indeed, the NLP functionality of SHAP was more effective, as we can see in figure 4.



```python
data = [
    ("Who was the founder of Constantinople?", "Constantinople was founded by Constantine."),
    ("What is the Haber process?", "The Haber process synthesizes ammonia."),
    ("What was Leonardo Da Vinci famous for?", "Leonardo Da Vinci was famous for his inventions.")
]

# Convert questions and contexts into input format
inputs = [q + " [SEP] " + c for q, c in data]

# Build an explainer using a token masker
explainer = shap.Explainer(predict, tokenizer)

# Generate SHAP values for each question-context pair
shap_values = explainer(inputs)

# Output SHAP values
for i, sv in enumerate(shap_values):
    print(f"SHAP Values for input {i+1}:")
    shap.plots.text(sv)
```

```
PartitionExplainer explainer:  33%|███       | 1/3 [00:00<?, ?it/s]

PartitionExplainer explainer: 100%|██████████| 3/3 [01:31<00:00, 25.06s/it]

PartitionExplainer explainer: 4it [02:21, 47.05s/it]SHAP Values for input 1:
```

base value
-5.98547
f(inputs)
5.64352
-4
-2
-0
2
4
was ? . the of by who was constantinople constantinople [SEP] founder founded constantine
inputs
who was the founder of constantinople ? [SEP] constantinople was founded by constantine .

```
SHAP Values for input 2:
```

base value
-5.96781
f(inputs)
0.829304
-5
-4
-3
-2
-1
0
ce EP ha ber is synth the process what es the ? izes ha ber . ammonia
inputs
what is the haber process ? [SEP] the haber process synthesizes ammonia .

```
SHAP Values for input 3:
```

base value
-5.99059
f(inputs)
1.4688
-5
-4
-3
-2
-1
0
1
c as da was for da vinci famous leonardo leonardo his . for famous ? what inventions
inputs
what was leonardo da vinci famous for ? [SEP] leonardo da vinci was famous for his inventions .

Fig. 4: SQuAD Model Interpretability Using SHAP

As we can see, SHAP provides more insight into how the model applies tokenization and reaches its most likely conclusion. While this may not fully alleviate concerns about model performance, we propose that model interpretability should be included in the test battery during model validation prior to production deployments of any model. Given the rate of improvement across the machine learning discipline, it is likely a matter of time before model interpretability packages are able to provide deeper insight into the model's reasoning, and potentially localize layers in the architecture that might be causing elevated errors.

For the purpose of MLOps integration, instead of running a separate model interpretability container, it may be more appropriate to incorporate checks using packages like SHAP into a model validation component. However, that possibility requires further analysis, and the optimal positioning of model interpretability within an MLOps lifecycle may ultimately depend on the organizational structure and requirements of the engineering team and larger enterprise. Therefore, flexibility in the implementation of model interpretability is likely preferable to proscriptions regarding its use.

III. Can ML pipelines in Kubeflow be triggered using GitHub Actions?

In order to determine if traditional DevOps practices such as CI/CD can be fully applied to MLOps, we wanted to determine if an MLOps pipeline run can be initialized from either a GitHub commit or using a cron job for scheduling. For an enterprise environment where there is continuous data being streamed (perhaps through Apache Kafka), or ingested routinely and either stored in a data warehouse or data lake, automating ML model training is a vital feature for an engineering team to deploy. Automation enables engineers to reduce their task saturation, and also mitigates single points of failure in an organization (if the sole engineer responsible for model retraining takes a vacation, for example). Therefore, in the interest of organizational resilience and employing best engineering practices, we integrated GitHub Actions into Kubeflow, with the resulting pipeline being initialized by a cron job, shown in figure 5.

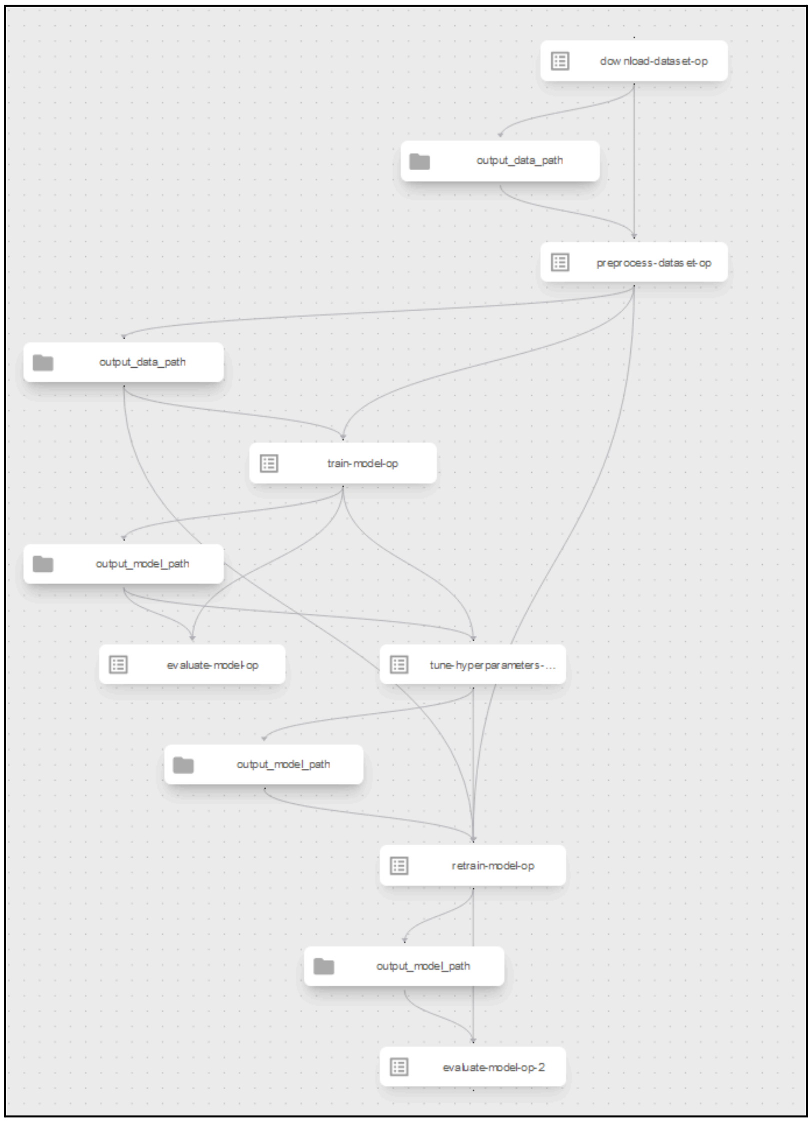


Fig. 5: Automated Pipeline Initialization with Kubeflow and GitHub Actions

In the interest of conserving compute resources, we did not let the full pipeline run, but it was successfully started. While it is possible that there might have been errors that we did not catch due to the early termination, it is likely that the pipeline would have executed successfully. As we can see, there was no difficulty in incorporating GitHub Actions into the MLOps process

IV. Is Kubeflow a viable MLOps platform to use at scale?

The major cloud providers (Google Cloud Platform, Amazon Web Services, and Microsoft Azure) have all developed MLOps capabilities as part of their offerings, but they all have additional costs associated with their use. Consequently, an open source offering like Kubeflow presented an interesting opportunity to reduce costs. For a medium or large enterprise regularly running machine learning pipelines, those costs could easily enter into the thousands of dollars, such that open source solutions provide interesting opportunities.

In terms of initial configurations, Kubeflow did not save time when compared with cloud provider offerings. While this project did not directly compare configuration time, past experience with Google's Vertex AI and Amazon's Sagemaker was considerably more efficient, faster, and less prone to errors. Furthermore, because Kubeflow is deployed within a specific region of a cloud provider, migrating the cloud infrastructure out of that region is impossible without rebuilding the entire Kubeflow environment, which could easily take multiple hours. Because Kubeflow utilizes Kubernetes for container orchestration, and the default Kubeflow node pool does not have GPU enabled, the engineer then has to deploy a different node pool in the same region with the desired compute capabilities. Unfortunately, due to cloud resource availability, that is not always feasible, such that it required multiple days (approximately 20 hours) to successfully deploy a Kubernetes node pool that had the required computational capabilities to train the ML models.

Once the cluster was configured, the ML training proceeded without issue, and the GitHub Actions and SHAP integration posed no challenge. However, because Kubeflow is open-source, some of its functionality isn't fully developed, such that some of the Katib algorithms did not function, especially when it came to Neural Architecture Search (NAS). Consequently, our ability to fully assess the AutoML performance was somewhat impaired, and moving the trained models from Kubeflow into a different environment wasn't as seamless as the organic cloud provider process.

Between the difficulties in the initial Kubeflow deployment, the resource allocation challenges, and the limited functionality of the platform, we determined that future MLOps research and work is best conducted within the organic technical stack of a cloud provider, as it enables broader integration, better documentation, and a greater rate of product improvement. Consequently, based on our experience, Kubeflow is not a viable MLOps platform for a mature engineering organization.

Deliverables

All of the deliverables are contained in this public repo:

https://github.com/msegal347/COMS_6156_Final_Experiment

Self-Evaluation

I found the overall research process illuminating, as it forced me to challenge the assumptions under which I approached the project. While I did some research into Kubeflow (and had the midterm experiment to do some exploring), the full limitations of the platform didn't become fully apparent until I was halfway into the process.

My overall intent behind the project was to assess Kubeflow as an MLOps platform, and determine if the incorporation of model interpretability, AutoML, and GitHub Actions would make for a fully functional, automated pipeline process. While I was able to complete the ML model training, the difficulties related to Kubernetes node pools and the region restriction made the process far more time consuming than I anticipated.

In the future, I intend to spend far more time on feasibility assessment, which I think would involve more extensive documentation review, consulting community forums, and watching tutorials. I first encountered difficulties during the midterm experiment, especially when dealing with outdated Kubeflow documentation, and should have considered that as a warning about the current state of the platform.

In this respect, I think a valuable lesson is that for future research, it is important to either use mature and fully supported platforms, or find experimental technology that has a lively and active open-source community. Because Kubeflow's development has been greatly slowed over the past few years, that probably should have served as a cautionary indicator.

Nevertheless, I found the experience valuable and highly productive. Not only did I gain deeper experience with different model architectures and datasets, but I also implemented multiple end-to-end ML pipelines, incorporated novel packages, and successfully accomplished the project goals I had set for myself. I believe I am much better positioned to evaluate future MLOps solutions, and to approach research in the area with a more seasoned perspective.

Appendix

I greatly enjoyed the course, and found that it contained the perfect combination of freedom and structure to pursue interesting research. I thought the milestone projects were a great way to stay on track, and I thought the paper presentations were immensely valuable in terms of consistent exposure to research practices, experimental design, and scientific writing.

I think a possible addition could be an “anticipated difficulties” or “potential points of failure” section in the project proposals, where students can discuss and formally brainstorm areas where their research projects might fail or run into difficulties. Additionally, developing a contingency plan in the event of failure could be an interesting section of the proposal, as it would pre-position students to pursue alternative lines of inquiry if a project proves to be non-viable.

Given the scope of the course these suggestions may not be practical, but they were the only ones I could muster for an otherwise phenomenal process.

References

1. Kubeflow, https://www.kubeflow.org/docs/

2. ImageNet Benchmark Performance,
https://paperswithcode.com/sota/image-classification-on-imagenet

3. SQuAD Benchmark Performance,
https://paperswithcode.com/sota/question-answering-on-squad20

4. ResNet-18,
https://huggingface.co/microsoft/resnet-18

5. BERT-SQuAD,
https://github.com/kamalkraj/BERT-SQuAD/tree/master

6. SHAP, https://shap-lrjball.readthedocs.io/en/latest/index.html

7. Google Kubernetes, https://cloud.google.com/kubernetes-engine/docs