Automating Web-based Infrastructure Management via Contextual Imitation Learning

Jieyu Lin*
Electrical and Computer Engineering
University of Toronto
Toronto, Canada
jieyu.lin@utoronto.ca

Hongxiang Geng*
Electrical and Computer Engineering
University of Toronto
Toronto, Canada
kevin.geng@mail.utoronto.ca

Alberto Leon-Garcia
Electrical and Computer Engineering
University of Toronto
Toronto, Canada
alberto.leongarcia@utoronto.ca

***Abstract*—Web-based management dashboards provide intuitive interfaces to support various infrastructure management tasks. However, the lack of programmability in these dashboards makes it difficult to automate management tasks. To address this limitation, we propose a system call MAIL (Management Automation through Imitation Learning) that learns from an operator's demonstrations using imitation learning to automate management tasks on web-based dashboards. A novel Contextual Imitation Learning algorithm is proposed in MAIL to overcome the learning challenges brought by the requirements of infrastructure management automation. The effectiveness of the MAIL system has been demonstrated using both synthetic environments and production environments. It outperforms existing methods and is able to achieve an 86%-100% success rate for most tasks.**

I. Introduction

With the rapid development of cloud/edge computing, big data, artificial intelligence, and web-scale applications, today's IT infrastructures are becoming ever more complex, which makes management more challenging and time consuming. Worst still, these infrastructures are continuously evolving to meet the fast changing needs of the applications. This adds more burden to the management team as they need to constantly adjust their strategies and develop new automation tools to meet the management requirements.

Infrastructure management teams today use either 1. command line interface or 2. Graphical User Interface (GUI) to monitor and manage various IT systems and resources. These two methods each have their pros and cons. Command line allows operators to develop customized scripts for automating some management tasks. However, it often requires the operator to have programming expertise. Besides, with the management requirements constantly evolving, it may not be cost efficient to develop an automation script due to high development overhead and limited payback. On the other hand, GUI provides an easy-to-use interface that allows operators to execute some management tasks without the requirements of programming skills. However, GUI also comes with its shortcoming, which is the lack of programmability compared to command line interface. This makes it challenging to automate management tasks using the GUI.

Is there a way to get the best of both worlds? In this work, we look into designing a system that provides intuitive programmability for web-based GUI by learning from an operator's demonstrations in order to automate infrastructure management tasks. The goal is to pass on the operator's management knowledge to machine learning models through demonstrations. There are a number of challenges related to designing such a system, including:

- **Limited demonstrations**: the system needs to learn from a small number of demonstrations and be able to automate the management tasks.
- **Long action sequences**: some automation tasks involve taking a long sequence of actions, which is known to be challenging for current learning algorithms.
- **Learning to infer decisions**: from the demonstrations, the system needs to learn and infer the management decision rules to be able to accurately automate the tasks.
- **Handling various data types**: the system needs to be able to handle different types of data including image, various charts, and text from the web browser in order to make proper action decisions.

To solve the above challenges, we developed MAIL (**M**anagement **A**utomation through **I**mitation **L**earning), a system that uses imitation learning to learn from an operator's demonstrations to automate web-based management tasks. Within the MAIL system, a new Contextual Imitation Learning (CIL) algorithm is proposed to overcome the aforementioned challenges of traditional learning methods for the setting of infrastructure management. Unlike other systems, our system is capable of processing multi-modal data from the browser (i.e. not only text information, but also images of the web browser screen). This allows our system to make decisions based on various visual contents (e.g. line chart, pie chart, and chart arrangement) during the automation process. To be able to focus on the right part of the screen, an attention mechanism is also added to our model. To the best of our knowledge, this is the first work on automating web-based infrastructure management tasks using learning methods.

We evaluated our system using 6 automation tasks on three dashboards: one custom-built dashboard with simulated data and two open-sourced dashboards running on a large-scale testbed with production data (OpenStack Dashboard and Grafana Dashboard). When compared to both existing heuristic methods and learning-based methods, our system is able to achieve much better performance in terms of successfully automating the management tasks. Furthermore, we have also

* equal contribution

©Copyright IEICE - APNOMS 2021

shown the extensibility of our system for continuous improvement and supporting new tasks through an extensibility study.

The main contributions of our work are as follows:

- We design and implement the MAIL system that automates web-based infrastructure management tasks using imitation learning techniques.
- We propose a novel Contextual Imitation Learning algorithm that is capable of handling long action sequence tasks and multi-modal browser inputs.
- We design a set of benchmarks for evaluating web-based infrastructure automation systems.
- We demonstrate for the first time using our MAIL system that learning methods can be used in web-based interfaces for automating infrastructure management. MAIL is able to achieve over 86% success rate for 5 out of 6 tasks.

II. Motivation Example

In this section, we provide a motivating example of the type of web-based management automation tasks that we consider in this work. Imagine an infrastructure management operator responsible for managing a list of Virtual Machines (VMs) using a web-based dashboard. These VMs are hosting web servers to serve web requests. The goal of the operator is to ensure the proper operation of the applications. The usual routine of the operator is to go through each VM's individual page to view its CPU and memory statistics. If the CPU utilization or memory usage is too high, then the operator will resize the VM to give it more CPU cores and memory. On the other hand, if the operator observes an abnormal pattern of the CPU utilization (e.g. sudden drop), the operator would bring up more relevant monitoring data charts such as network IO and disk usage to help zoom in to the problem. He/she then generates a report of the issue and also tries to take an action (e.g. reboot) on the VM attempting to fix the issue.

This example demonstrates a number of important infrastructure management tasks, such as ad-hoc dashboard construction, anomaly detection, reporting, and management action execution. To automate such a management routing, the automation system first needs to be able to process the aforementioned various charts (e.g. CPU utilization line chart, etc), and learns to classify normal and abnormal conditions. The system also needs to choose the right browser action at each step of the automation, such as going through each VM one by one. The number of browser actions increases linearly with the number of VMs, which makes it a challenging learning task for traditional learning methods.

III. Background and Related Work

A. Infrastructure Management

There is a large body of work in the area of infrastructure management. Open-sourced systems such as OpenStack, OpenNebula, Kubernetes provide capabilities to allow management of different resources such as VMs, containers, and networks. Other systems such as Grafana, InfluxDB, and Prometheus provide monitoring and visualization capabilities to facilitate infrastructure management. Large testbeds such as SAVI[1], GENI[2], and CloudLab[3] are developed to support the management and orchestration of future infrastructures and applications. Cloud providers such as AWS and Azure offer software interfaces to help the management of cloud resources. One commonality shared among these systems is that they all provide web-based user interfaces for infrastructure management purposes. However, none of them provides programmability through their web interface, making it difficult for non-programmers to automate management tasks.

B. Web Browser Automation

Researchers have been looking into providing solutions to automate web browser tasks. CoScripter [4] is one of the earlier works in the area, where web browser automation processes are manually recorded using a browser plugin and these "scripts" are shared among the users. Sikuli [5] uses screenshots as input and allows the users to program automation procedures using the screenshots. Although these methods may work for some simple fixed scenarios, they are not suitable for infrastructure management tasks as the monitoring data can be constantly changing. Learning-based methods have also been proposed to automate web browser tasks. WGE [6] and DOM-Q-NET [7] propose using Reinforcement Learning (RL) for automating web browser tasks. WGE uses expert demonstrations to regularize the exploration of the RL algorithm. DOM-Q-NET takes natural language instructions as input and processes the HTML DOM tree using Graph Neural Network (GNN) to produce browser actions. As these two methods need to process the whole DOM tree from the web browser, their performance degrades as the complexity of the DOM tree increases, which limits their applications in real-world scenarios. These methods are also unable to process image and chart inputs as the contents are not visible in the DOM tree. Furthermore, none of the mentioned previous work has looked into supporting web browser automation for infrastructure management tasks.

IV. Method

In this section, we focus on discussing the imitation learning method used in MAIL to support management automation. We first describe the high-level overview of the method and then go into the detailed model design.

A. Method Overview

To automate network management tasks using our system, the operator first interacts with the browser as usual to execute the management task to record the demonstrations. Then these demonstrations are used to train a Contextual Imitation Learning model which learns to take a browser action (e.g. click, scroll) based on the browser input (i.e. screenshot, browser text) to accomplish the automation task. Once the model is trained, it is able to execute the task automatically without needing help from the operator. During execution time, at each time step, the model basically takes inputs from the browser and makes an action decision (e.g. click at a certain location). Once the action is taken, we wait until the browser finishes loading/reacting. Then we move to the next time step, and the model repeats until the task is completed.

To be able to support long action sequences and multi-modal inputs (screenshot & text) from the browser for infrastructure

©Copyright IEICE - APNOMS 2021

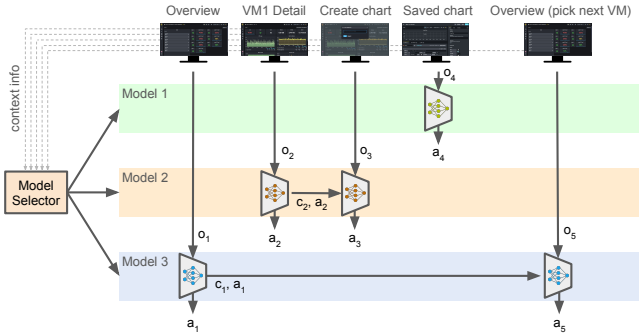


Fig. 1: Contextual Imitation Learning method overview. Notations are defined as a: action, o: observation (image and text), c: context. The subscripts represent the time steps.

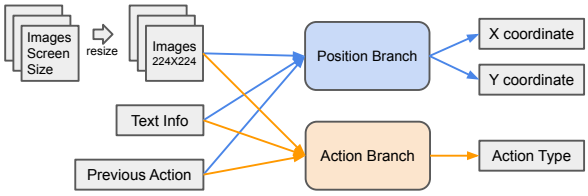


Fig. 2: Model Overview

management tasks, we designed a new learning method called Contextual Imitation Learning (CIL), which breaks down the management tasks into multiple sub-tasks and selects an appropriate model based on the context (e.g. browser URL). Figure 1 illustrates the idea of this method. Time steps that belong to the same task use the same model and each task maintains and passes along their context variables and past actions, so they can quickly pick up where they left off. An analogue of this is Context Switch in Operating Systems where the state of the process is backed up before context switching and restored after. Such a design can reduce the length of actions for each task and make it simpler to handle a long sequence of actions and hierarchical tasks, such as going into each VM to conduct some sub-tasks. Currently we use the URL as the input for the model selector to decide which sub-task the current time step belongs to. Our method can also support more advanced model selection methods (e.g. image-based) for more complex scenarios.

B. Model Overview

Next we describe the design of our learning model with an overview followed by detailed descriptions. Figure 2 provides an overview of the model. The inputs of the model consist of browser screenshot image, text information from the browser, and previous action (e.g. cursor position). These inputs are fed into both a Position branch and an Action branch to generate the required output for action execution. The Action branch is responsible for making the decision regarding the type of action to take. Currently it supports Left/Right Click, Wait, Drag, and Scroll actions. The Position branch is responsible for outputting the additional position information required to support the action, such as x,y coordinates for click position and y coordinate for the length of the scroll action.

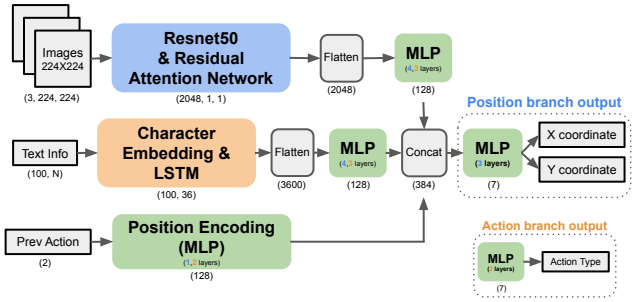


Fig. 3: Detailed Model Design.

C. Detailed Description of Action and Position Branch

We use a similar architecture for the Action branch and the Position branch. The detailed model is shown in Figure 3. We use three different types of sub-networks to processing the multi-modal inputs (image, text, actions). Then we combine their output features through concatenation followed by a Multi-Layer Perceptron (MLP). The details of the sub-networks are described below:

- **Image sub-network**: To process screenshot images from the browser, we use the Resnet50 Convolutional Neural Network (CNN) architecture due to its good performance in image processing. However, in our setting, instead of just looking at the whole screen, we want the model to also be able to give attention to important areas or charts on the screen. Thus we added a residual attention mechanism inspired by [8] to our network, which improves the overall performance.
- **Text sub-network**: To process the text presented on the screen of the web browser, we perform a character-level embedding for the texts and pass them to a Long Short-Term Memory (LSTM) unit for aggregation. We specifically chose character-level embedding instead of word-level embedding because of the characteristics of the infrastructure management dashboards. Many dashboards contain non-regular words such as IP addresses, VM names, etc. Given their large variety, it is non-trivial to learn the embedding of all these words. Thus character-level embedding is more suitable in our scenario.
- **Previous action sub network**: This network is responsible for processing the previous action to give more context to the model.

V. Implementation

In this section, we describe the detailed implementation of our MAIL system. Figure 4 shows the implementation diagram of the MAIL system. At a high level, the system consists of a web browser module and a cloud module. The web browser module runs locally in the operator's computer and is responsible for supporting demo recording, and executing the automated management tasks launched by the operator. The cloud module is responsible for storing demo data, training, and storing imitation learning models.

To use this system for automating a management task, the operator first goes through the training phase. He/she uses the Demo Recorder component to record several episodes

©Copyright IEICE - APNOMS 2021

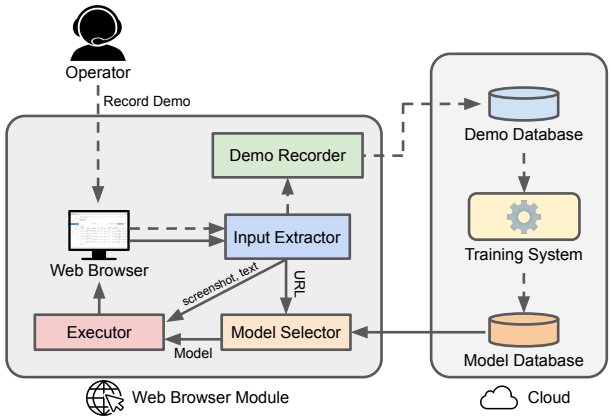


Fig. 4: Implementation Diagram of our CIL system. Dash line: training pipeline; solid line: execution pipeline

of executing the task (Recorded episodes from past working experience can also be used). These episodes are sent to the cloud and stored in the Demo Database. The Training system then trains Contextual Imitation Learning models and stores them in the model database. This completes the training phase. When the automated task is launched, the input data is obtained by the Input Extractor from the web browser and a model is chosen by the Model Selector based on the URL. These together are used by the executor to pick a management action for the web page. If the automation task is not completed, these modules repeat with the new input data from the web browser, until the task is completed.

Next, we describe the details of each component:

Training System: We used the Pytorch library to build our Training System. For each training step, we pick a batch of random image and text pairs from the demo database. We preprocess the screenshot image inputs by resizing it to (3, 224, 224). Different loss functions are used for the two branches of the model. The Action branch uses Cross Entropy Loss while the Position branch uses MSE Loss function.

Model Selector: This component maintains a key-value table with URL regex and model name. During execution, it matches the browser URL to the table to obtain the model name. The corresponding model is then retrieved from the Model Database (or local model cache) and returned to the Executor. To avoid excessive remote requests to the Model Database, the Model Selector maintains a local model cache of the models used so most requests can be fulfilled locally.

Executor: The Executor uses the model supplied by the Model Selector and the information extracted by the Input Extractor to get the encoded action. Post-processing is performed on the encoded action to look up the action type and to recover the actual pixel location. The Executor then uses Selenium to send the action to the browser.

Input Extractor: At each timestamp, the Input Extractor obtains 4 types of data: URL, screenshot image, presented text and cursor position. We use Selenium to obtain the first three types of data and PyAutoGUI for the cursor position.

Demo Recorder: The Demo Recorder is used for recording demos, and can be started and stopped by the operator. Once started, it automatically records the actions that the user takes in the browser such as Click, Drag, Scroll, etc. The recorder stores the actions along with the corresponding data from the Input Extractor to the Demo Database for training purposes.

VI. Evaluation

A. Experiment Description

To evaluate our system's performance, we design benchmarks using three network management dashboards: 1. **Custom Cloud Management Dashboard**; 2. **Openstack Dashboard**; 3. **Grafana Dashboard**. In each environment, we design two automation tasks.

*1) **Custom Cloud Management Dashboard**:* The Custom Cloud Management Dashboard is a network monitoring and management dashboard we built for VM management. The dashboard displays an overview of the VMs and enables the administrator to easily control the machine based on monitoring data. Some functionalities it provides include monitoring CPU and memory utilization, starting, stopping, and resizing the machines. Refer to Figure 5a for a screenshot of the dashboard. A detailed page of each VM is accessible by clicking the name of the VM. This dashboard is built using Python with the Django library. It is built based on the requirements of cloud network management, and it enables easy customization to better evaluate our system under different scenarios. We designed 2 tasks for this environment:

1. **Failure Detection & Restart (FDR)**: For each machine that shows an inactive status in the overview panel, the operator needs to start the machine by opening the dropdown menu in the ACTIONS column and selecting the start instance option.
2. **Memory Overload Detection & Resize (MODR)**: This task involves going into the detailed view of each VM from the overview page. Then based on a memory usage pie chart, the operator needs to resize the VM (i.e. adding more memory) if the memory usage is above 75%.

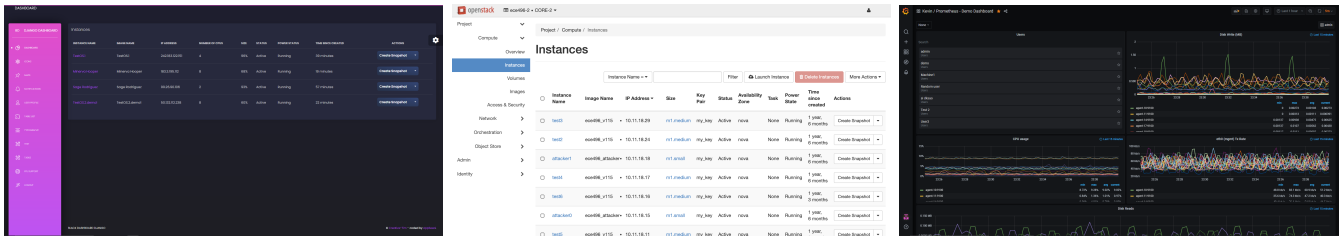
*2) **Openstack Dashboard**:* Figure 5b shows the Openstack's Horizon dashboard we used to evaluate our method. OpenStack is widely used in many production environments. In our case, we use the production Openstack dashboard of SAVI testbed. The data in this dashboard are all real production data. The 2 tasks for this environment are:

1. **Single-Page Failure Detection & Restart (SFDR)**: For each stopped VM, the operator needs to click the corresponding start button in the dropdown menu and wait for it to start.
2. **Multi-Page Failure Detection & Restart (MFDR)**: For each stopped server, the operator needs to first click the name of the server to go to the detailed view. Then start the server by clicking the top-right start button. It also needs to wait for the server to finish the starting process.

*3) **Grafana Dashboard**:* Grafana is another well-known dashboard for infrastructure monitoring. This dashboard provides highly customizable features that enable the operator to easily add information and visualize them in various forms including charts, numbers, and texts. We use Grafana with Prometheus database containing production monitoring data of physical machines from the SAVI testbed. Based on the features, we designed two automation tasks.

1. **Report Generation (RG)**: Click the physical server names sequentially and generate each server's current status report

©Copyright IEICE - APNOMS 2021



(a) Custom Cloud Management Dashboard (b) OpenStack Dashboard (c) Grafana Dashboard

Fig. 5: Screenshots of three dashboard environments used for evaluation

by clicking a report generating button.

2. **Overload Warning & Investigation (OWI)**: Go to the dashboard of each server that has a CPU usage above 100%. Add a panel using the CPU utilization information for that machine and save the dashboard.

B. Comparison Methods

For evaluation, we compare our method to both heuristic-based methods and learning-based methods. For heuristic-based methods, we compare to an action memorization method and a demo replay method. These two methods are variants of the techniques described in the previous work [4]. For learning-based methods, we use behavioral cloning, which is a type of imitation learning. The details of the methods are:

- **Action memorization** (heuristic-based): This method processes the demonstration data to construct a database of what next action to take given the previous action and cursor location. During execution, it looks up the database based on the current cursor position and previous action and picks the next action.
- **Demo Replay** (heuristic-based): This method randomly samples a demo from the demonstration database and replays the actions recorded in the demo.
- **Behavioral Cloning** (learning-based): This method uses the behavioral cloning learning method [9] to make action decisions. A Resnet50 network is used with inputs of image and previous action. A single model is trained to make decisions for all time steps of one task.

C. Overall Performance

We evaluate our system using the 6 tasks described above in Section VI-A. We measure the success rate of our system on each of these tasks. The success rate is defined as the ratio of successfully completed test runs versus the total number of test runs. A test run is considered as successfully completed if the model is able to accomplish the goal of the automation task. For evaluation, the operator collects 10 demos for each task. Then each method executes 100 runs on the same test dataset we collected for evaluating their success rate. Table I shows the evaluation success rate of our method and the compared methods. We can see that our method is able to achieve a high success rate for most tasks. The heuristic-based methods (i.e. Action Memorization and Demo Replay) failed in many tasks, especially in the production Openstack Dashboard. This suggests that manually specifying action sequences by the operator does not work well for web-based infrastructure management automation tasks, especially

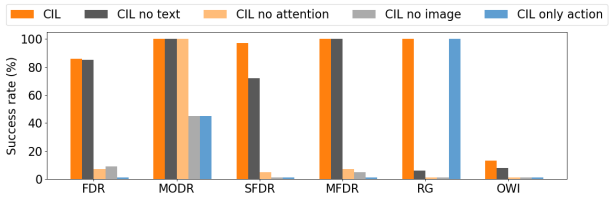


Fig. 6: Ablation Study

in real-world environments. This is because the monitoring data and the content on the dashboard during execution time may not be exactly the same as the demonstration, so learning becomes crucial in this case to be able to generalize and make accurate action decisions. The learning-based Behavioral Cloning method performs better than the heuristics-based methods. However, it is still unable to achieve a high success rate for several tasks such as the OpenStack Dashboard tasks. We believe this is due to the Behavioral Cloning method being unable to handle the long sequence of actions. Although the model is trained to select accurate actions (e.g. click at a location), it may have small errors during each step of the execution, which adds up quickly as the number of actions increases. On the other hand, our Contextual Imitation Learning method breaks down the task automatically into sub-tasks, so the sequence of actions is not too long for each task. Also by processing multi-modal input data from the browser, our method is able to further improve its performance. This is discussed further in the Subsection VI-D below.

D. Ablation Study

To evaluate how different features of our system contribute to the overall performance, we conduct an ablation study by removing some features of the system.

For all 6 automation tasks, we compare the original Contextual Imitation Learning algorithm with: 1. CIL without image branch (i.e. removing the blue branch in Figure 3); 2. CIL without attention mechanism in the image branch; 3. CIL without text input branch (i.e. removing the yellow branch in Figure 3); and 4. CIL with only past action branch (i.e. only keeping the bottom green branch in Figure 3). Figure 6 shows the success rate of these methods. From these results, we can observe that the attention-based image processing of the browser screenshot has the highest impact on the success rate of the automation tasks. The screenshot not only provides visual information for the chart but also the overall structure of the dashboard. Next, CIL without text branch shows perfor-

©Copyright IEICE - APNOMS 2021

TABLE I: Success Rate

Environment	Task Name		Methods			
			Action Memorization	**Demo Replay**	**Behavioral Cloning**	**MAIL (Our method)**
Cloud Management Dashboard	FDR	Failure Detection & Restart	7%	8%	9%	**86%**
	MODR	Memory Overload Detection & Resize	0%	2%	**100%**	**100%**
Openstack Dashboard	SFDR	Single-Page Failure Detection & Restart	0%	0%	7%	**97%**
	MFDR	Multi-Page Failure Detection & Restart	0%	0%	7%	**100%**
Grafana Dashboard	RG	Report Generation	**100%**	**100%**	99%	**100%**
	OWI	Overload Warning & Investigation	0%	4%	8%	**13%**

mance degradation compared to the CIL method, which shows the importance of having text processing capability for these automation tasks. Although the texts can be seen on the screen of the browser, it is non-trivial to train the image branch of the model to extract all the necessary text information accurately to make proper decisions. Overall, this ablation shows the importance of having a multi-modal processing pipeline for the automation of web-based infrastructure management.

E. Extensibility Study

1) Model extension for continuous improvement: One type of extensibility we aim to provide in our system is the ability to improve over time as we have more data. An operator should have the option to provide more demonstrations over time to update or improve an automation task. This allows the operator to pass on new knowledge, specify new rules, or improve the model's performance for specific scenarios. To evaluate this, we first train the system for the Failure Detection & Restart task with a fixed number of VMs. Then we add an additional VM to serve a new application, and ask the operator to provide 5 more demos. The system's success rate results are shown in Table II. We see that CIL performs well before adding the VM, but its success rate decreases as a new VM is added. After new demos are provided by the operator, the retrained model is able to quickly learn the automation task from the operator and achieve a high success rate. In fact the new demos allow the model to better infer the operator's intention, and even improve the performance of the original task.

TABLE II: Success rate for FDR and extended FDR tasks

Setup	CIL	retrained CIL
FDR task	86%	91%
Extended FDR task (+1 VM)	10%	87%

2) Extensibility for new automation tasks: Ideally, given the multi-model design of the MAIL system, the system should be able to leverage existing learned models in new tasks without having to train from scratch. To evaluate whether our system can support such an extension, we evaluate the system on a new task that is a hybrid of two known tasks by mixing models trained separately for each of the tasks. Specifically, we try to automate the task of restarting all the VMs by combining the first model from Memory Overload Detection & Resize and the second model from Failure Detection & Restart. We found that this hybrid model is in fact able to complete the new task with a high success rate. This shows the system's ability to support model mixing extension. One potential future work and extension is to identify and categorize which of the existing models can be used to solve part of a new task. With such extension, one can imagine eventually having a model database that can enable fast construction of models for automating required tasks based on the operator's demonstrations.

VII. Conclusion

In this work, we proposed the MAIL system that automates web-based infrastructure management tasks by learning from the demonstrations provided by the operators. With a learning method called Contextual Imitation Learning (CIL), we show that the system can automate various management tasks by only learning from a small number of demos. We hope our work can inspire further work in this area to further improve automation of web-based infrastructure management.

References

[1] J.-M. Kang, T. Lin, H. Bannazadeh, and A. Leon-Garcia, "Software-defined infrastructure and the savi testbed," in *International Conference on Testbeds and Research Infrastructures*. Springer, 2014, pp. 3–13.

[2] "Geni." [Online]. Available: https://www.geni.net/

[3] D. Duplyakin, R. Ricci, A. Maricq, G. Wong, J. Duerig, E. Eide, L. Stoller, M. Hibler, D. Johnson, K. Webb *et al.*, "The design and operation of cloudlab," in *2019 {USENIX} Annual Technical Conference*, 2019.

[4] G. Leshed, E. M. Haber, T. Matthews, and T. Lau, "Coscripter: automating & sharing how-to knowledge in the enterprise," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, 2008.

[5] T. Yeh, T.-H. Chang, and R. C. Miller, "Sikuli: using gui screenshots for search and automation," in *Proceedings of the 22nd annual ACM symposium on User interface software and technology*, 2009, pp. 183–192.

[6] E. Z. Liu, K. Guu, P. Pasupat, T. Shi, and P. Liang, "Reinforcement learning on web interfaces using workflow-guided exploration," in *International Conference on Learning Representations*, 2018.

[7] S. Jia, J. R. Kiros, and J. Ba, "Dom-q-net: Grounded rl on structured language," in *International Conference on Learning Representations*, 2018.

[8] F. Wang, M. Jiang, C. Qian, S. Yang, C. Li, H. Zhang, X. Wang, and X. Tang, "Residual attention network for image classification," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017.

[9] M. Bojarski, D. Del Testa, D. Dworakowski, B. Firner, B. Flepp, P. Goyal, L. D. Jackel, M. Monfort, U. Muller, J. Zhang *et al.*, "End to end learning for self-driving cars," *arXiv preprint arXiv:1604.07316*, 2016.

©Copyright IEICE - APNOMS 2021