Towards Hybrid Automation by Bootstrapping Conversational Interfaces for IT Operation Tasks

Jayachandu Bandlamudi[1], Kushal Mukherjee[1], Prerna Agarwal[1], Sampath Dechu[1], Siyu Huo[1], Vatche Isahagian[1], Vinod Muthusamy[1], Naveen Purushothaman[2], Renuka Sindhgatta[1]

[1]IBM Research
[2]IBM Consulting

Abstract

Process automation has evolved from end-to-end automation of repetitive process branches to hybrid automation where bots perform some activities and humans serve other activities. In the context of knowledge-intensive processes such as IT operations, implementing hybrid automation is a natural choice where robots can perform certain mundane functions, with humans taking over the decision of when and which IT systems need to act. Recently, ChatOps, which refers to conversation-driven collaboration for IT operations, has rapidly accelerated efficiency by providing a cross-organization and cross-domain platform to resolve and manage issues as soon as possible. Hence, providing a natural language interface to bots is a logical progression to enable collaboration between humans and bots. This work presents a no-code approach to provide a conversational interface that enables human workers to collaborate with bots executing automation scripts. The bots identify the intent of users' requests and automatically orchestrate one or more relevant automation tasks to serve the request. We further detail our process of mining the conversations between humans and bots to monitor performance and identify the scope for improvement in service quality.

Introduction

Until recently, the primary purpose of automation was to automate the highly repetitive tasks in a business process that were executed manually by the users (Syed et al. 2020). However, the tasks most suitable for automation had an enclosed cognitive scope and were susceptible to human error (van der Aalst, Bichler, and Heinzl 2018). This led to a horizontal segmentation of processes where certain processes are fully automated while others would be entirely manual. However, in recent years, researchers have been exploring a more hybrid approach, which considers vertical segmentation of processes. This involves human-robot collaborations where there is a hand-off of tasks between humans and robots, collaborating to achieve higher efficiencies (Cabello Ruiz et al. 2022). Further, certain industries still require human-in-the-loop automation for many reasons such as compliance, risk, and liability (Rizk et al. 2020).

IT operations (IT-Ops) is one such industry where there is a great push toward compliance, traceability, and approval flows. IT automation tools have been increasingly utilized to meet service level agreements (SLAs) and optimize metrics related to mean time to repair (MTTR) in these complex environments. Around 41 percent of enterprises use ten or more IT performance monitoring and automation tools (Adnan Masood 2019). A few automation tools such as Ansible[1] provide developers with a way to create and host automation (scripts) on various aspects such as (1) building, provisioning, and managing infrastructure, (2) application deployment, (3) network management, (4) report generation, and other similar tasks. These tools also, provide easy-to-use interfaces where site reliability engineers (SREs) may explore the catalog for automation and launch them as and when required with the required inputs. However, a user often needs to decide and trigger automation, hence the need for a human in the loop.

Given the need for user intervention to address errors, exceptions, and rollback within IT automation, a recent paradigm shift in IT Operations has been the adoption of ChatOps (Hand 2016). ChatOps facilitates interactions between different SREs using a conversational interface to gather information, help perform tasks or guide towards the incident resolution. ChatOps reduces recovery time for high-priority incidents, increases transparency and knowledge management by providing visibility into common information on the chat interface, and provides improved system security using role-based access control (Tobey 2016). More importantly, ChatOps offer the perfect platform for hybrid automation enabling systems where bots and humans co-exist and collaborate by sharing knowledge and handing over tasks. The primary mode of interaction between bots and humans on ChatOps is the natural language. This enables seamless handover of tasks between humans and bots and enables vertical segmentation of processes.

Given the landscape of IT automation tools, the gap that exists is the encapsulation of IT automation into a form that can exist in the ChatOps environment, interacting with users through natural language. In some cases, developers create custom/ad-hoc bots adding conversational interfaces that are expensive to develop and hard to maintain. The resulting bot sprawl (due to the multitude of automation tools) in ChatOps is also disconcerting to users (Davenport 2019). Further,

[1]https://www.redhat.com/en/technologies/management/ansible

many existing deployed IT automation scripts in multiple enterprise domains can be leveraged by enabling a conversational interface. This paper describes a framework to add a conversational interface to an existing automation service, enabling a seamless hybrid automation platform agnostic to the underlying automation platform.

The main contributions of the paper are (1) Enabling a conversational interface for IT automation agents by the bootstrapping framework that significantly reduces development effort, (2) Presenting an approach for asynchronous collaboration between humans and IT automation agents (from multiple automation platforms) via ChatOps, (3) Assessing the effectiveness of the implementations by monitoring the mined event logs from conversations between human agents and IT automation agents. Identifying new automation opportunities from the mined conversation logs.

System Architecture

Several automation platforms such as Ansible and Terraform are used to automate repetitive IT operation tasks such as user access management, network management, infrastructure management, application maintenance, and report generation. In a typical IT process with automation, there are three types of users: (i) ***Automation developers*** are responsible for developing task-specific automation scripts or playbooks using automation platforms, (ii) ***Site Reliability Engineers (SRE)*** are responsible for deciding and running appropriate automation for resolution and maintaining systems, and (iii) ***Automation users*** are the end users facing issues and want them to be resolved.

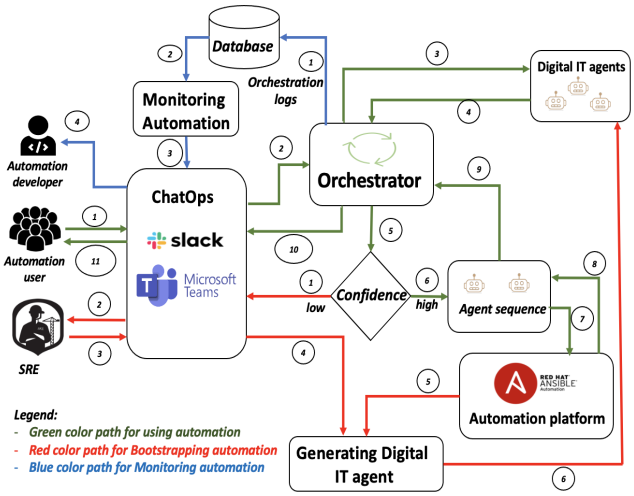

Figure 1: System Architecture

The standard automated IT process starts with an automation user creating ad-hoc requests via an email or a ticketing tool, and then SREs act on them by running one or more automation scripts. There are several disadvantages to this approach. Firstly, request resolution take longer time due to dependency on SREs, secondly, automation users need to ensure that the correct SRE group is assigned for their request, and a significant training effort is associated with acquiring this knowledge. Lastly, there are numerous automation platforms and SREs need specialized expertise in different automation platforms to run automation.

We aim to address the challenges mentioned above by enhancing accessibility for automation users to run automation, improving collaboration between users, and intelligent sequencing of automation. For this purpose, we use conversational interfaces that are familiar to the automation users (such as Teams[2] or Slack[3]), using which they collaborate and use the natural language to run automation for resolving issues and managing IT systems. However, in the ChatOps environment, creating several conversational interfaces for each IT application and each automation would lead to a bot sprawl. The recommended way to manage such rapid addition of bots is to consider generalized frameworks or tools that can perform a variety of automation (Davenport 2019).

To alleviate this issue, we utilize a framework that builds digital assistants (Rizk et al. 2020). The authors proposed a framework that bridges the gap between RPAs and enterprise chatbots. It uses a multi-agent orchestrator that enables interactive automation through natural language and AI planning. Next, we present the core concepts used in the framework (Rizk et al. 2020).

- ***Skills:*** refer to the atomic functions. there exist three types of skills in the framework: (i) *understand* user's requests, (ii) *act* on the request, and (iii) *respond* to the user. The *understand* skill refers to the natural language understanding of a user's utterances. The *act* skill, in our context, is the IT automation scripts. Finally, the *respond* skill refers to skills that provide a human-consumable response based on the output of the *act* skill.
- ***Agents:*** An agent is a composition of *understand-act-respond* skills that transform the set of skills into a conversational entity. Agents receive a user utterance and the current context of the execution or state. They act on the utterance based on the context and respond with an updated context and confidence in their response, indicating their comprehension of the input and response.
- ***Orchestration:*** The orchestration allows cooperation between agents, that include *human* agents. The main function of an orchestrator would be to i) select the correct agent from a predefined set of available agents to achieve a task, ii) manage the context, and iii) manage the flow of the conversation between the agents.

  The orchestrator receives an utterance and forwards it to all agents available along with appropriate context from the user. Once it receives the agents' preview response, it chooses the most appropriate agent (or sequence of agents) to execute that utterance after selecting the appropriate agent. It connects with the agent to complete the execution.

The system architecture (Figure 1) shows automation users running automation by providing a natural language utterance as a message in the ChatOps (path 1 in Figure 1).

[2]https://docs.microsoft.com/en-us/graph/teams-concept-overview

[3]https://api.slack.com/

Based on the user utterance, the orchestrator chooses the appropriate IT agent required to run (path 2). Compared to the standard process where SRE acts on user requests, in our system, the orchestrator is the intermediary and proactively listens to user requests in ChatOps. The orchestrator also identifies appropriate digital agents, asynchronously runs automation (path 3-10), interactively asks users to provide inputs at run-time (path 11), and notifies users about execution status.

If the orchestrator cannot find an appropriate IT agent for the given user utterance, the SRE is notified via ChatOps and requested to choose the appropriate automation. This initiates the Digital IT agent bootstrapping framework i.e. a new IT agent is created for the user-requested automation, and the orchestrator will run the new Digital IT agent detailed in *Bootstrapping Framework* section (the path for bootstrapping in Figure 1). The system stores conversational interactions as orchestration logs in a database. The Monitoring Automation module helps the automation developers monitor automation via ChatOps, to realize the automation value and find new opportunities for automation (detailed in *Monitoring Automation* section).

Bootstrapping Framework

SREs leverage the bootstrapping framework to generate conversational interfaces for existing automation on the automation platform. The red color path in system architecture Figure 1 shows SREs bootstrapping automation by a generating digital IT agent. Automation platforms provide an interface to run automation scripts. For example, the Ansible platform provides an Ansible Tower interface that enables users to run the automation repetitively by creating a job. Each job template includes several attributes such as the playbook name (automation script), the automation description, the inventory (host server), the variables required by the playbook, and the credentials needed to run the playbook.

To integrate ChatOps with automation scripts, we need to create its corresponding skills and agents. In this section, we describe how we create the skills that are the building blocks of the agent configuration. Figure 2 highlights the flow of our generation of skills. Given an automation script, we create three core skills: *understand*, *act*, and *respond* by utilizing the information available in the *automation job template* or the automation scripts.

Configuring the Understand Skill: The purpose of the *understand* skill is to receive the user utterance and evaluate the likelihood that the agent would be able to serve the user's request. In a typical chat-bot application, the *understand* skill comprises an intent classifier trained using a set of sample (or training) utterances (Qi et al. 2020). However, for the domain of IT automation, these sample utterances are not available. Hence, our approach is to parse the description of the automation and extract the key phrases. Subsequently, the key phrases are used to generate sample utterances that a user could say to invoke the automation. These sample utterances are then used to train an intent classifier.

Step1 - Extract key phrases from the description: Many approaches exist for key phrase extraction from short texts,

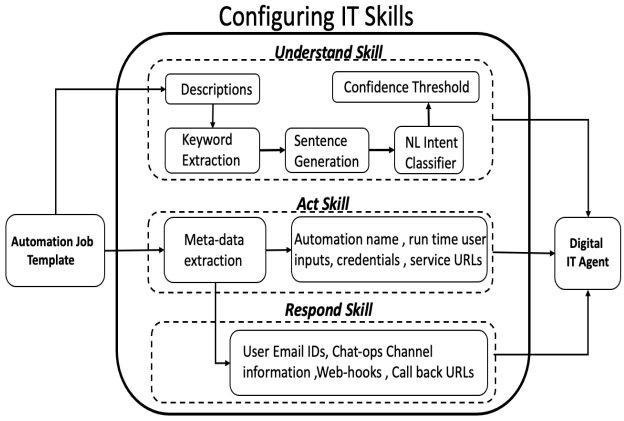

Figure 2: Flow to generate Digital IT Agent

such as Dependency Parsing (Honnibal and Johnson 2015) and Abstract Meaning Representation (Banarescu et al. 2013) to extract verb-noun phrases. We propose to use KeyBert[4] which generates a list of key phrases that are most similar to the text description, thereby capturing the gist of the description. We generate the key phrases from the description field in the metadata for a given automation script. If more than one key phrase is generated, we choose the one with the highest score.

Step2 - Generation of sample utterances: Upon receiving a key phrase from the keyword extraction module, we utilize a fine-tuned T5 model for paraphrasing the key phrase into complete sentences that a user may say to invoke the automation. As we do not have a domain-specific dataset for paraphrasing, the T5 model (Raffel et al. 2020) is fine-tuned by using the Quora[5] dataset, which consists of a set of duplicate questions from their website. The supervised training data-label pair of the form ($"Paraphrase :< sentence_i >"$, $" < paraphrase_i"$) are created from the duplicate questions, which are then used to train the T5-model. For the generation of paraphrases, the key phrase obtained from the keyword extraction module is converted into a sentence using boilerplate templates (for eg. "Can you help me {KeyPhrase}"). The resulting sentence is prefixed with the prompt "$Paraphrase :$ " and fed to the fine-tuned T5 model. Finally, beam search (Meister, Vieira, and Cotterell 2020) is used to generate multiple paraphrases of the given input sentence.

Step3 - Train an intent classifier: We use the above step for each automation script to generate paraphrases of utterance that the user may say to invoke it. All these sentences are labelled with the intent corresponding to that automation script. The intent classifier is trained with this labelled data, and it serves as the *Understand* skill in the agent associated with corresponding automation. The intent classifier predicts a probability score(confidence) for the intent

[4]https://github.com/MaartenGr/KeyBERT

[5]https://quoradata.quora.com/First-Quora-Dataset-Release-Question-Pairs

based on input utterance. Recent studies (Qi et al. 2021; Liu et al. 2019) provide a comparison of various intent classifier applications for intent classification. In this work, we use RASA (Bocklisch et al. 2017) as our intent classifier. However, other tools such as Watson Assistant, LUIS & Haptik (Qi et al. 2020) can be integrated into our framework.

Step4 - Adaptive confidence threshold selection: The orchestrator selects an appropriate agent based on the confidence from *understand* skill. However, the orchestrator will not select the agent if the confidence score is below a certain threshold. Hence it is crucial to specify a valid confidence threshold for the orchestrator to work. Our system dynamically manages the confidence threshold for the intent classifier. We use a trained intent classifier, validation data for the intent classifier, and out-of-domain data to identify the appropriate confidence threshold value. The two publicly available intent classification data sets that are used for out-of-domain samples are *ATIS* dataset (automated airline travel inquiry systems) and *HWU64* dataset (conversations between humans and digital assistants)(Qin et al. 2021).

The confidence threshold is computed adaptively. Suppose there are N intent classes in the training dataset. We construct a validation dataset by augmenting the sentences with sentences from the out-of-domain data sets to give us $N+1$ intent classes (where the $(N+1)^{th}$ intent class corresponds to "out-of-domain" intent). Given a sample, the output of the intent classifier is one of the original intent classes $n \in 1, 2, ..., N$ along with a confidence score for the class c_n. If the confidence threshold is $\lambda \in [0, 1]$, the modified class label is computed as

$$Class\ label = \begin{cases} n, & \text{if } c_n \geq \lambda \\ N+1, & \text{otherwise} \end{cases}$$

We now obtain predicted labels for the entire dataset and compute *F1-score*. The choice of the threshold λ is made by using a grid search to maximize the *F1-score*Our system uses one intent classifier for each domain, such as ERP and Ticketing systems. We retrain the intent classifier whenever new automation is bootstrapped and the new confidence threshold is computed.

Configuring the Act Skill: *Act* skill is responsible for running user-requested automation. This skill is configured using the meta-data extracted by parsing the job templates. Automation platforms provide multiple fields to specify meta-data such as automation name, run-time user inputs, credentials, service URLs etc. These details are required to run automation with proper access and on the desired server by consuming user inputs. Figure 3 (a) shows the *Act* skill for running Ansible automation with user inputs.

Configuring the Respond Skill: The purpose of *Respond* skill is to notify users about the status of the automation request. Automation platforms support various mechanisms to send responses to users in the form of Emails, Web-hooks etc. which are specified as meta-data. We parse and use this information to create *Respond* skill. Figure 3 (a) shows *Respond* skill to get the automation job status and notify users.

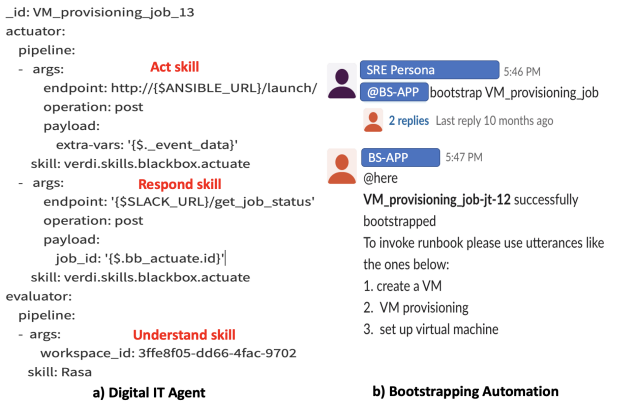

Figure 3: Bootstrapping Automation

Generation of a Digital IT Agent

As mentioned previously, a digital IT agent consists of a pipeline of one *Understand* skill, one or more of *Act*, and one or more *Respond* skills. Figure 3 (a) shows generated Digital IT agent, which is framework native no-code file generated using multiple skills, where *Act* skill is used to run an automation script on Ansible platform, *Respond* skill is used to notify the status of automation execution to users on the slack application and *Understand* skill is the conversational interface that matches user utterance with automation description. Fig 3 (b) shows SRE bootstrapping conversational interface for automation, and framework generates one digital IT agent for corresponding automation. It is to be noted that Bootstrapping framework is integrated with the ChatOps using a custom application that is built using Slack APIs [6]. Using the custom application SRE can automatically generate a digital IT agent with all required skills using command **bootstrap ⟨automation job name⟩**, this command can be used to bootstrap conversational interfaces for various automation across different platforms.

Orchestrating Multiple Digital Agents

When the orchestrator (Rizk et al. 2020) receives a natural language request from the user, it broadcasts it to all the available agents. The agents then use their respective *understand* skill to return confidence or serve that request. The orchestrator normalizes all confidences to arrive at a score for each agent. Next, the orchestrator selects the top 1 (or top K) agents with the highest scores (above a minimum threshold). Then the orchestrator sequences the execution of the *act* and *respond* skills for the selected agents as described in (Rizk et al. 2020). Once a digital IT agent is generated it can be invoked by user utterance via ChatOps (such as Slack or Teams).

Monitoring Automation

An important aspect of IT automation is to monitor the value obtained using digital IT agents and it can be measured in

[6] https://api.slack.com/

terms automation coverage and time savings. In this section, we design a method that uses orchestration logs from the deployed system database. Orchestration logs are asynchronous hybrid conversations between digital agents and humans that are pre-processed to generate an event log required for further analysis.

In the pre-processing, each end-to-end conversation goes through (i) The associated resource (user and agents) extraction, (ii) Turn filtering using regular expressions, to discard trivial utterances such as greetings and chit-chat.

The processed event log is utilized to compute several insights, such as the number of user automation requests handled by digital agents in both success and failure scenarios, compute the time digital agents take to run automation, and recommend new automation opportunities to the automation developers.

Topic Clustering

To recommend new automation opportunities, topic clustering is performed on the conversations in the event log to analyze the type of automation user requests the current agents cannot handle. The steps are given below:

- Each user request utterance undergoes linguistic preprocessing to filter out stop-words and perform lemmatization using the python NLTK (Loper and Bird 2002) library. Also, key phrases are extracted from the preprocessed utterance using Key-Bert.
- The key phrases are ranked based on the scoring function that calculates the importance of each candidate key phrase using the *Inclination Factor* (IF). IF calculates the *Informativeness* or inclination of a key phrase to an utterance. To determine the global inclination of a key phrase, the IF of each label is determined as the average of the IF of a key phrase with each utterance.
- IF is calculated as the average of linguistic inclination (LI) and semantic inclination (SI). Where LI is determined using Jaccard similarity between each key phrase with utterances. SI is determined using the Cosine similarity between the sentence embeddings(Devlin et al. 2019) of key phrases and utterances.
- The candidate key phrases are ranked based on the IF scores. The k top key phrases forms the clusters and the utterances that contain a key phrase forms a cluster.

The Monitoring Automation module is integrated into the system as a native digital agent, and automation developers use ChatOps to get insights related to automation monitoring and new opportunities.

Evaluation of System Performance

The proposed system is deployed to five customers on IT automation use-cases related to ERP system and Ticketing tool. As shown in Table 1 the *No.of classes* corresponds to different IT automation, in the ERP deployment we have 70 different IT automation related to system monitoring, configuration, report generation, and user maintenance tasks. The Ticketing tool deployment contains 9 different IT automation for ticket creation, updating, group assignment, and report generation. The *No.of conversations* represents the number of conversation threads initiated by the automation users, extracted from the ChatOps. The *Agent-invoking utterances* represent the utterances that invoked a digital agent. The *No.of automation descriptions* represents the developer-provided description for the automation in the automation platform, multiple descriptions are specified for each automation. The *Testing set samples* are the number of user utterances that were collected from ChatOps. The labels for these utterances were supervised via a crowd-sourcing effort within our enterprise. Below, we evaluate the performance of the intent classifier, which is critical for enabling a conversational interface for automation. Subsequently we present automation monitoring results to show benefits of bootstrapping automation.

Dataset	ERP	TICKET
No.of classes	70	9
No.of conversations	11320	15430
Agent-invoking utterances	9665	11342
No.of automation descriptions	195	24
Testing set samples	7465	9240
Client deployments	3	2

Table 1: Datasets description

Classifier	**Dataset**	**Acc.**	**F1**	**Mean Conf.**	**Conf. threshold**
SVM	ERP	0.48	0.48	-	-
SVM	ERP-SG	0.91	0.90	-	-
SVM	Ticket	0.73	0.71	-	-
SVM	Ticket-SG	0.84	0.84	-	-
RF	ERP	0.48	0.49	-	-
RF	ERP-SG	0.65	0.66	-	-
RF	Ticket	0.57	0.55	-	-
RF	Ticket-SG	0.75	0.75	-	-
RASA	ERP	0.82	0.85	0.74	0.65
RASA	ERP-SG	**0.93**	**0.91**	**0.94**	**0.85**
RASA	Ticket	0.86	0.78	0.93	0.85
RASA	Ticket-SG	**0.91**	**0.90**	**0.99**	**0.95**

Table 2: Intent classifier results

Intent Classifier

Table 2 shows three intent classifiers evaluated using two metrics *Accuracy* and *F1-score*. RASA (Bocklisch et al. 2017) classifier is deployed as a docker container, and *train*, *predict* APIs were used to train the classifier and to obtain predictions. On the other hand, SVM and Random Forests(RF) classifier implementations from scikit-learn (Pedregosa et al. 2011) were trained using tf-idf (Ramos 1999) features with 5-fold cross-validation to find optimal hyper-parameters.

Results are shown for the four data sets used to train the intent classifiers. The *ERP* and *Ticket* data sets include only the automation descriptions and serve as the baselines. On the other hand, the *ERP-SG* and *Ticket-SG* data sets represent the deployed system where descriptions are processed through the keyword extraction and sentence generation pipeline. Hence ***SG*** data sets contain amplified set of samples. Moreover, *ERP* training set has 195 samples,

whereas *ERP-SG* training set has 3986 samples. Similarly, the *Ticket* and *Ticket-SG* training sets have 24 and 525 samples, respectively.

The *Accuracy* and *F1-score* reported in Table 2 are obtained for the testing set. The *ERP* testing set has 7465 samples and the *Ticket* testing set has 9240 samples. We also report the Mean confidence of the predictions on the entire testing set and computed adaptive confidence threshold. It is observed that using the train set with samples from the sentence generation pipeline improved the Accuracy, F1-score for SVM, Random Forests classifiers, and even *Mean confidence* of predictions by a good margin. Our deployed system uses RASA(Raffel et al. 2020) intent classifier trained using automation descriptions amplified using key word extraction , sentence generation pipeline.

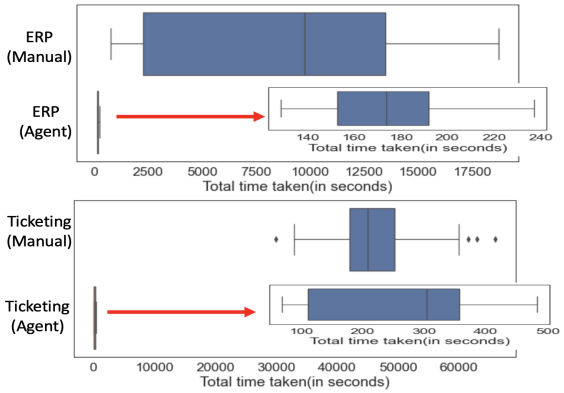

Figure 4: Time savings

Monitoring Automation

To monitor the value of automation, we try to empirically answer the following monitoring questions -

Q1) How often are agents invoked?: From Table 1 for ERP deployment out of the 11320 total conversations, the agents were invoked for 9665 utterances i.e., for 85% of the conversations. Similarly, for *Ticket* deployment, total conversations are 15430, and agent invoking utterances are 11342, i.e., for 73% of the conversations.

Q2) How often is the correct agent invoked successfully?: From Table 2, for ERP deployment, correct agents are invoked with 96% accuracy, and for *Ticket* deployment, it is 92% accuracy.

Q3) How much time is saved when the correct agent is invoked?: We compare the median time taken by digital agents against the time taken by humans to perform the task. From Figure 4, for ERP deployment, digital agent's median time is around 3 minutes, whereas the median time taken by human agents is approximately 2.5 hours. Similarly, for *Ticket* deployment, the digital agent's median time is around 5 minutes, whereas the median time taken by human agents is approximately 12 hours.

Q4) What are the new automation opportunities for bootstrapping?: For the ERP deployment, out of the total of 11320 conversations, there were 1665 conversations (i.e., 11320-9665) where an agent wasn't invoked (see Table 1) and similarly for *Ticket* deployment there were 4088 such conversations. These uncovered conversations were addressed by the SREs directly and analysing them helps to determine the new agents for bootstrapping such that the automation coverage is improved. We use the topic clustering described in the previous section to analyze the uncovered conversations. After analyzing the significant clusters, we found that the agents to handle the following requests were missing: email reports, server deletion, change login protocol, user migration, server connectivity issues, and cloud server configuration. We can improve the automation coverage by bootstrapping new agents to handle these requests.

Deployment and Maintenance

At present, the system is deployed in the cloud environment on the Openshift platform[7] as a multi-tenant service with five separate docker images, each handling different functionalities. Several of these services are integrated into the system through REST APIs developed using FastAPI[8] framework. The first docker image caters to the orchestrator, which handles multiple digital IT agents. The second docker image handles interaction between ChatOps & orchestrator, python language-specific Slack and Teams APIs are used for developing this functionality. The third docker image is the RASA (Bocklisch et al. 2017) intent classifier which provides APIs for training and predicting. The fourth docker image is the MongoDB database, which stores the entire system's conversational data. Finally, the fifth docker image is for monitoring automation. Decoupling functionality into several images helps manage different system parts separately. We have a development team of 20 people who manage this product following an agile development methodology. Maintenance also includes deploying monthly releases with new enhancements and bug fixes in the production environment. For this purpose, we have our DevOps pipeline to build, test and deploy.

Conclusion and Future Work

Bootstrapping conversational interfaces for automation comes with several benefits, such as less turnaround time for user requests , reduced development effort. Our framework offers scalability to integrate different automation platforms and handle bot sprawl (Davenport 2019). Also, the deployed system can monitor the automation execution to identify new opportunities. However, the deployed systems' performance relies on the intent classifiers' accuracy. Model classification errors could lead to the possibility of executing incorrect automation. Hence, it is required to add support for rollback skills to alleviate the consequences of running incorrect automation. Additionally, as trust is an essential factor required while running correct automation, the system should explicitly ask for user approval to run automation the first few times to gain confidence.

[7]https://www.redhat.com/en/technologies/cloud-computing/openshift

[8]https://fastapi.tiangolo.com/

References

Adnan Masood, A. H. 2019. *Cognitive Computing Recipes*. O'Reilly Media, Inc. ISBN 97814842410595.

Banarescu, L.; Bonial, C.; Cai, S.; Georgescu, M.; Griffitt, K.; Hermjakob, U.; Knight, K.; Koehn, P.; Palmer, M.; and Schneider, N. 2013. Abstract Meaning Representation for Sembanking. In *LAW@ACL*.

Bocklisch, T.; Faulkner, J.; Pawlowski, N.; and Nichol, A. 2017. Rasa: Open Source Language Understanding and Dialogue Management. *CoRR*, abs/1712.05181.

Cabello Ruiz, R.; Jiménez Ramírez, A.; Escalona Cuaresma, M. J.; and González Enríquez, J. 2022. Hybridizing humans and robots: An RPA horizon envisaged from the trenches. *Computers in Industry*, 138: 103615.

Davenport, T. H. 2019. How to Tame "Automation Sprawl". *Harvard Business Review*, July: 103615.

Devlin, J.; Chang, M.-W.; Lee, K.; and Toutanova, K. 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *Proceedings of the 2019 Conference of the NAACL*, 4171–4186. Minneapolis, Minnesota: Association for Computational Linguistics.

Hand, J. 2016. *ChatOps*. O'Reilly Media, Inc. ISBN 9781491962305.

Honnibal, M.; and Johnson, M. 2015. An Improved Non-monotonic Transition System for Dependency Parsing. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, 1373–1378. Lisbon, Portugal: Association for Computational Linguistics.

Liu, X.; Eshghi, A.; Swietojanski, P.; and Rieser, V. 2019. Benchmarking Natural Language Understanding Services for building Conversational Agents. In *IWSDS*.

Loper, E.; and Bird, S. 2002. NLTK: The Natural Language Toolkit. *CoRR*, cs.CL/0205028.

Meister, C.; Vieira, T.; and Cotterell, R. 2020. Best-First Beam Search. *Transactions of the Association for Computational Linguistics*, 8: 795–809.

Pedregosa, F.; Varoquaux, G.; Gramfort, A.; Michel, V.; Thirion, B.; Grisel, O.; Blondel, M.; Prettenhofer, P.; Weiss, R.; Dubourg, V.; Vanderplas, J.; Passos, A.; Cournapeau, D.; Brucher, M.; Perrot, M.; and Duchesnay, E. 2011. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research*, 12: 2825–2830.

Qi, H.; Pan, L.; Sood, A.; Shah, A.; Kunc, L.; and Potdar, S. 2020. Benchmarking Intent Detection for Task-Oriented Dialog Systems. *CoRR*, abs/2012.03929.

Qi, H.; Pan, L.; Sood, A.; Shah, A.; Kunc, L.; Yu, M.; and Potdar, S. 2021. Benchmarking Commercial Intent Detection Services with Practice-Driven Evaluations. In *NAACL*.

Qin, L.; Xie, T.; Che, W.; and Liu, T. 2021. A Survey on Spoken Language Understanding: Recent Advances and New Frontiers. arXiv:2103.03095.

Raffel, C.; Shazeer, N.; Roberts, A.; Lee, K.; Narang, S.; Matena, M.; Zhou, Y.; Li, W.; and Liu, P. J. 2020. Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer. *Journal of Machine Learning Research*, 21(140): 1–67.

Ramos, J. 1999. Using TF-IDF to Determine Word Relevance in Document Queries.

Rizk, Y.; Isahagian, V.; Boag, S.; Khazaeni, Y.; Unuvar, M.; Muthusamy, V.; and Khalaf, R. 2020. A Conversational Digital Assistant for Intelligent Process Automation. In *BPM 2020 Blockchain and RPA Forum, 2020, Proceedings*, volume 393 of *LNBIP*, 85–100.

Syed, R.; Suriadi, S.; Adams, M.; et al. 2020. Robotic Process Automation: Contemporary themes and challenges. *Comput. Ind.*, 115: 103162.

Tobey, A. 2016. Incident Management and Chatops @ Netflix Feat Scorebot. Santa Clara, CA: USENIX Association.

van der Aalst, W.; Bichler, M.; and Heinzl, A. 2018. Robotic Process Automation. In *Bus Inf Syst Eng 60*, 269—272.