A Hybrid Crowd-Machine Workflow for Program Synthesis

ABSTRACT

Despite advances in machine learning, there has been little progress towards creating automated systems that can reliably solve difficult tasks, such as programming or scripting. In this paper, we propose techniques for increasing the reliability of automated systems for program synthesis task via a hybrid workflow that augments the system with input from crowds of human workers. Unlike previous hybrid workflow systems, which have been focused on less complex tasks that crowd workers can do in their entirety (e.g., image labeling), our proposed workflow handles tasks that untrained crowd workers cannot do alone (i.e., scripting). We show that we can improve the performance of an automated system by integrating crowd workers into targeted portions of the task workflow. We evaluate our approach by creating BashOn, a system that increases the accuracy of an automated program that generates Bash shell commands from natural language descriptions by ∼30%.

ACM Classification Keywords

H.5.m. Information Interfaces and Presentation: Misc.

Author Keywords

program synthesis; crowdsourcing; crowd workflows

INTRODUCTION

The Bash shell is a complex but powerful user interface. Users can parameterize and combine Bash commands to quickly perform complex operations that would take much longer in a Graphical User Interface (GUI). However, correctly writing a Bash script command requires complex reasoning skills and years of practical experience. Even experienced Bash users often need to rely on support from a variety of resources like Stack Overflow or man pages [22, 23] for recalling keywords, function names, argument types, and methodologies.

Although researchers have created automated systems to help users write Bash scripts and programs from natural language descriptions [25, 18, 8], they are limited to accomplishing constrained tasks. In our tests with Tellina [18], a state-of-the-art automated tool to translate natural language query into Bash commands, we found a 10% accuracy rate. Part of this is because currently, most automated systems only work reliably in domains where there is sufficient, well organized training data [7] and the "output" of the system is simple [21]. The only reliable source for support in complex domains like Bash scripting is from human experts, who can be difficult to find.

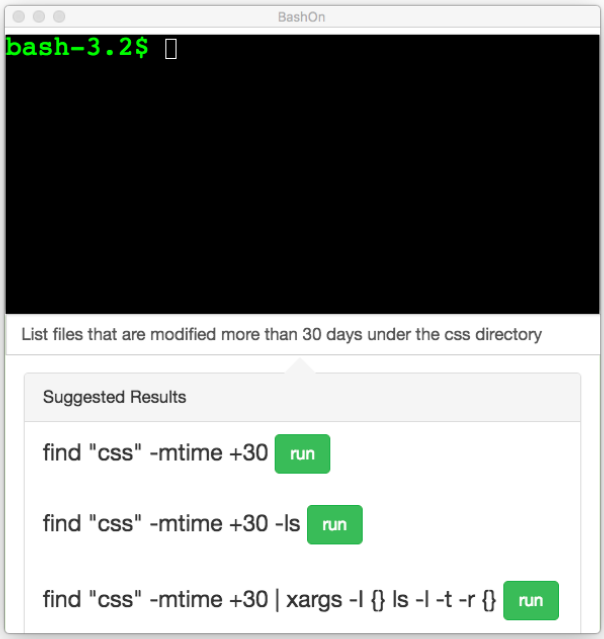

Figure 1. BashOn allows users to create Bash commands from natural language. It uses a hybrid workflow that integrates crowd workers and an AI for program synthesis.

In this paper, we build a hybrid workflow for generating Bash commands that augments an existing program synthesis system—Tellina [18]— with non-expert crowd workers. We show that despite not having expertise with Bash scripting, crowd workers can increase the effectiveness of the fully-automated system by nearly 30%. We also introduce BashOn (Fig. 1), a user interface that uses our hybrid workflow to allow users to generate Bash commands from natural language descriptions.

An important feature of BashOn is that it relies on crowd workers who do not have experience with Bash commands. This is in contrast with previous hybrid workflow systems, which rely on crowd workers' ability to perform a task in its entirety [3, 17, 16, 14, 6]. Thus, when using non-expert crowd workers, these systems are limited to domains that are relatively low complexity (i.e., little prior knowledge is required). A key insight when designing BashOn is that although non-expert workers are less effective than an automated system at generating Bash commands, they can be effective in targeted portions of its workflow. For example, although a given crowd worker might not be able to determine what a given Bash command does, they can still help the system by determining whether the *result* of a candidate matches the users' intentions. They can also support the system by leveraging their contextual understanding to further assist in specifying commands.

We explored multiple ways to restructure the hybrid workflow and then we compared their performances. Our experiments found that although the crowd struggled at times with pieces of domain-specific information, overall they were able to make up for places where the automated system was lacking. Although the focus of BashOn is on generating Bash commands, the ideas behind its workflow could apply to Artificial Intelligence (AI) in many complex domains. This paper contributes:

- Methods, challenges, and opportunities in implementing the hybrid workflow in a program synthesis task.
- Evidence that integrating crowd workers into automated system workflows improves performance on complex tasks.
- BashOn, an interactive system that instantiates this idea of a hybrid workflow in the domain of script programming.

RELATED WORK AND PROBLEM BACKGROUND

Our work builds on prior work in hybrid intelligence, crowd workflows, and program synthesis.

Hybrid Intelligence Workflows

Crowdsourcing can be used to improve machine learning by performing tasks that automated systems cannot do well (e.g., entity extraction and filtering [26, 20]), validation of machine-generated results [5], and creation of new training data from which the automated systems can learn [14, 6]). These systems show the promise of hybrid workflows, but they rely on crowd workers' ability to perform a task in its entirety. Systems like Tohme and Zensors [12, 15] leverage both machine learning and crowdsourcing techniques to augment performance in different applications. Tohme uses this combination to detect curb ramps in Google Street View, while Zensors creates sensor feeds that can respond to natural language questions. In addition, studies have shown that hybrid workflows that interleave human labor and machine intelligence perform better than either component alone [27, 2]. However, none of these systems require any domain-specific knowledge from crowd workers in order for them to participate. Our work introduces a new, hybrid intelligence workflow that leverages off-the-shelf program synthesis tools to avoid the need for worker expertise for an otherwise expert task.

Program Synthesis and Programming Support

Programming is a difficult task that requires years of training and practice. Even for experienced programmers, memorizing every syntactic rule is nearly impossible, so they have to rely on existing resources. Many community question answering (CQA) websites, like Stack Overflow [22], provide online asynchronous support that allows software developers to post questions to a large community. But these CQA sites have a number of limitations, such as long wait times to receive an answer after posting the requests, and the fact that answers are not personalized [19]. Other work has used human experts to provide support [4], but finding the right expert can be prohibitively difficult and expensive.

There has been extensive research on how to efficiently synthesize programs from natural language descriptions. Gulwani et al. conducted a series of studies to synthesize spreadsheet scripts by developing Domain Specific Languages (DLSs) and synthesis algorithms to learn user-provided examples [11, 10]. But DLSs are not for programmers to read but for machines, and they are domain-dependent. Systems like Codemend [25] and Tellina [18] aim to translate natural language queries to programs or scripts using automated systems trained with expert-generated data. But the amount of data that is available to train these systems is still small, which limits the model performance. Our hybrid workflow requires automated modules to operate the translation, but there is no need for a human expert to supervise the process. Furthermore, other automated systems often produce complex programs that are different from what programmers usually write, and may require non-trivial manual effort to validate and troubleshoot. BashOn, instead, leverages human intelligence to validate the program execution outcomes, which we show to be an effective way of improving its performance.

Tellina: Synthesizing Bash Commands

To explore the effectiveness of using hybrid workflows for generating Bash commands, we built on Tellina [18], a state-of-the-art fully automated system for synthesizing Bash commands. Tellina's automated workflow consists of three primary steps (illustrated in Figure 2):

- *Entity Extraction:* This first step finds all of the entities and types in the natural language query. This parser restricts input to context-independent queries (e.g., no 'this', 'it'), and often requires users to refine queries using special characters (e.g., putting "" around names, strings, and regular expressions).
- *Language Translation:* This step takes the entity extracted query template and translates it into a ranked list of possible Bash command templates using a Recurrent Neural Network (RNN) (sequence-to-sequence) model. Both the list of templates and the entities extracted from the first step are then forwarded to the Argument Filling step.
- *Argument Filling:* The final step fills the argument slots in the candidate commands with the extracted entities to form the complete script. Users then receive a list of full commands from this step. However, this step does not ensure that commands are safe, valid, or executable.

Performance

The reported accuracies of Tellina are $Acc_F^1 = 30.0\%$, and $Acc_F^3 = 36.0\%$ where Acc_F^k is denoted as the percentage of their test examples for which a correct full command is ranked k^{th} or higher in a list of possible translations. According to the authors, 41 out of 50 incorrect sample commands are caused by the mis-recognition of entities in the first step. In addition, the generated commands need to be validated by the users, and these commands can sometimes be unusual and complex, which makes validation difficult. Furthermore, this process could be very tedious and error-prone when using a large-scale file system.

These performance limitations provide an opportunity to introduce a hybrid workflow as a possible solution. Where Tellina's parser fails to extract the correct entities from a query, people

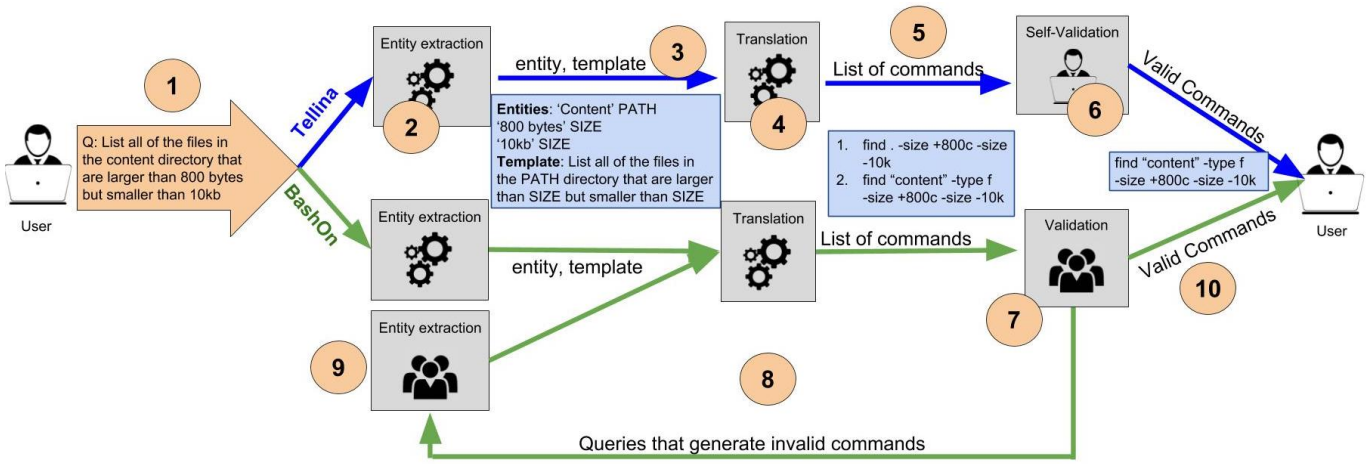

Figure 2. System diagram: This diagram plots both the original Tellina (blue) and BashOn's (green) workflow. In Tellina's workflow, first, a natural language query (1) is sent to an automated entity extraction process (2) which outputs a list of entities, their associated types, and the entity extracted query template (3). Then, the neural network (4) translates the query and extracted entities into a list of commands (5) that is sent back to the user (6). In BashOn's workflow, the natural language queries first go through the original Tellina system, with automated entity extraction (1-5). The results from this are then evaluated by crowd workers (7), and if the they are incorrect (8), are then go to crowd workers for entity extraction (9). Commands resulting from this re-extraction process go back to crowd workers for a final validation step (7), and the results of this are sent back to end-users (10).

could potentially be more accurate. Crowd workers could also help to validate the outcomes by voting on the execution results, as troubleshooting is a cumbersome process when the file system is large. However, the language translation and argument filling steps require domain-specific knowledge of the syntax of Bash commands, which non-expert crowd workers might not be able to gain in a short amount of time. So, for the translation step, we can continue to leverage Tellina's automated system.

THE BASHON WORKFLOW

The BashOn workflow consists of three steps: entity extraction by the crowd, automated language translation, and outcome validation by the crowd. The language translation part is inherited from Tellina. Here, we will explain how we designed and implemented the other two components. Our system architecture (Fig. 2) is similar to Tellina's; however, we replace two automated modules with crowd modules. We then hypothesize that these two integrated modules can each enhance the performance of Tellina, and lead to even more of an improvement when both are used together.

Entity Extraction by the Crowd

Prior work has shown promising results on leveraging crowd workers for name-entity recognition tasks [24, 13]. With regard to scripting tasks, understanding the context and semantics of a query is crucial to extracting the right entities. Thus, our first module utilizes non-expert crowd workers to extract entities from a natural language query. Workers are asked to write down all of the entities in the query and select their associated types. After computing the majority-voted entities and extracting them from the query, the query template is sent to be translated into the ranked command templates.

Outcome Validation by the Crowd

Our second module asks crowd workers to assist in validating commands that Tellina has returned. Normally, developers need to validate the commands from Tellina manually, which presents a few issues: Tellina gives no guarantee that the commands are executable, and sometimes the commands can be risky or have unintended side effects if the developers do not carefully review them (e.g., contains 'rm'). Thus, users are often frustrated by the syntactic errors present in the generated commands. Tellina also sometimes suggests unusually complex commands (e.g., long pipelines), and their user study participants found them "distracting even if some of these commands were correct" [18].

We instead ask crowd workers to validate and filter candidate commands by determining if the outcomes of these commands match the end-user's intention. For commands that require validating a large number of results (such as commands that are run on a large-scale file system), we decompose the verification step into small portions and distribute the reduced verification tasks to different crowd workers. Each worker is asked to use the given file system information to either reject or not reject the outcome of each command.

EVALUATION

To evaluate the effectiveness of our approach, we tested three conditions to examine the accuracy of each of the two crowd input modules, individually and together as BashOn, compared with the accuracy of using Tellina alone (Table 1).

Dataset

We adapted the queries used in Tellina's original experiment—real queries selected from online resources (e.g., Super User)—by having researchers rewrite them to avoid the correct answer from being found online through a keyword search. However, we only used queries that do not aim to modify the file system or any of its contents, as our outcome validation step requires us to run the candidate commands on an actual file system in parallel. Ten queries were left after filtering, and to increase the validity of our task set, we recruited two Bash users from Upwork (2+ years of Bash command experience) to rephrase these 10 queries in different ways (e.g., Fig. 2), as if they were

	S1: Tellina (baseline)	S2: Entity Extraction by the Crowd	S3: Validation by the Crowd	S4: BashOn
Accuracy/ Precision (Top 1)	10% (n=50)	28.00%* (n=50)	10% (n=40)	38.29**% (n=28)
Accuracy (Top 3)	17.28% (n=46)	45.71%** (n=46)	21.38% (n=29)	71.43**% (n=21)
Precision (Top 3)	11.11% (0.27,n=46)	32.38%** (0.39,n=46)	13.79% (0.50,n=29)	46.03**% (0.29,n=21)
Accuracy (Top 10)	18.60% (n=44)	51.52%** (n=44)	-	-
Precision (Top 10)	10% (0.22,n=44)	23.03%* (0.27,n=44)	-	-

Table 1. Accuracy is the percentage of queries that had a correct answer in the Top k commands returned, and precision is the percentage of the Top k commands returned that were correct. S3, S4 had fewer than 10 commands returned after filtering. n is the number of queries left at the end. Compared to baseline: * significant at $p < .05$; ** significant at $p < .01$

asking the questions themselves. After this, we had 50 queries in total.

Condition 1: Tellina (Baseline)

To establish a baseline, we entered all 50 queries into Tellina and computed the accuracy of its entity extraction (Table 2), as well as the precision of the top 1, 3, and 10 results (Table 1). The correct entities were extracted by two authors independently, and as there can be more than one entity in each query, we computed both the precision and recall. Note that the accuracy of Tellina [18] reported by Lin et al. was higher than it is in our experiment because our set of queries has looser constraints than the set that they used[18]. The commands are judged based on their execution outcome (e.g., list of file names), as there could be many correct commands and different representation of the outcome so they are difficult to validate automatically. The correct execution outcome is then compared with the correct output that was manually generated by the researchers.

Condition 2: Crowd-Powered Entity Extraction

We divided the 50 queries into 10 HITs, with each HIT containing five queries. To reduce any potential bias, we randomized the queries to minimize the chance that a worker would receive a rephrased version of a query they have already seen. Although prior studies like [13] have used the crowd to extract entities, it was unclear to us how well the crowd would be able to understand the task, as many of the queries contained domain-specific information. Thus, we iterated on this task by asking the crowd to provide feedback. The issues that were brought up the most were confusing definitions for each entity, and too many tasks in one Human Intelligence Task (HIT), as we had seven per HIT at the beginning. We then clarified the definition of each entity type in the instructions and only released five queries per HIT to increase worker accuracy and decrease mental fatigue.

In the final setting, we provided definitions of each entity type, two detailed examples, and graphical instructions on how to use our interface. We recruited 100 crowd workers (99% acceptance rate) where each crowd worker was asked to extract all possible entities for five queries, and we had ten workers parse each query. They spent about 45 seconds (S.D. = 30.11) on average, and we paid $0.24 for each query. For each task, there were 1.86 (S.D. = 0.61) entities on average. After removing far-off answers (e.g., non-existent word in the query), we computed the majority votes (6+/10). Table 2 shows the precision and recall where we found that the non-expert crowd outperforms the automated process from the original workflow. Also, the accuracy of the top 1, 3, and 10 answers all significantly increased by approximately 18%, 28%, and 33% compared to the baseline condition.

	Tellina	Crowd
Precision	29.00%	**85.99**%***
Recall	22.64 %	**91.99**%***

Table 2. Entity extraction precision and accuracy (* p<.001).**

Condition 3: Crowd-Powered Outcome Validation

The file system we used for this study is the same one that the original Tellina user study used (61 files, 4 level deep), as it is large enough to allow us to examine how well our validation strategy worked. We then divided the file system into three portions with each containing 20 (or 21) files, as we found that the workers are able to handle this amount well in our preliminary study. To reduce redundant work, we added a filter into our workflow before the execution results were sent to the crowd. We used both heuristics and static analysis [1] to filter out obviously error-prone commands (e.g., those that generated errors) before we sent them to crowd workers.

We had nine workers vote on each outcome (three per file system portion), and each worker was given both the query, the outcome list, and one portion of user's file system. Workers were asked to either disapprove (if they found evidence from the given file system that the command outcome was incorrect) or not disapprove (if they could not find such evidence) of each outcome. Then, the outcomes that have not been disapproved by any crowd workers are the correct ones.

We created 180 tasks (18 HITs, 3 assignments each), with each containing 10 commands' execution results and their original queries. We repeated each of these tasks for each portion of the file system that led to 540 tasks. There were on average 9.06 candidate commands per query (453/50), and workers spent 23 seconds (S.D = 12.44) on average on each one. We paid workers $0.13 for each validated execution result, for a total of $1.30 per HIT.

We then took the majority vote per file portion and aggregated the votes from all three portions. Together with the filter, this validation component was able to correctly validate 77.04% of commands, as shown in Table 3. The performance of this study is on par with the baseline condition, and we found that a portion of the queries were filtered out at the end as the crowd identified their results invalid.

Condition 4: BashOn

Finally, to evaluate how both crowd components perform jointly, we used the list of queries we generated in our second study (S2) and ran them through BashOn's full workflow, which is illustrated in Figure 2. Queries are first run through

	Automated filter	Crowd validation	Total
Accuracy	273/453 (60.3%)	76/180 (60.8%)	349/453 (77.04%)

Table 3. Accuracy of validation test in BashOn workflow. The automated filter only filters commands that generate error messages when executed.

automated entity extraction and translation components, as it would save effort if they could generate correct results, and the results from this are validated by the filter and crowd workers. If any of the possible outcomes are incorrect, the query is sent back to the entity extraction step, this time performed by crowd workers. The results from this extraction are used again to translate the query with the automated component, and are then re-validated by workers.

After this re-validation, we are left with two sets of potentially correct commands. We chose to sort this list first by the entity extraction method (Tellina's automated system, or crowd workers), and then by the number of crowd workers that voted to validate a command (with the max being three). We rank the commands that had entities extracted by crowd workers higher because of the better performance according to the results of our second study.

We used the same number of workers for each part as described in the previous two studies. Table 1 shows the accuracy and precision of the top 1 and 3 answers after aggregation. Overall, we found that BashOn outperformed all the other three workflows and significantly increased the accuracy by 40% and 30% in the top 1 and 3 answers respectively. We also noticed that not all queries had answers returned by BashOn due to the low translation accuracy, and BashOn's final answers for some queries consisted of less than 10 candidate commands due to our filter and crowd validation.

DISCUSSION

Challenges and Solutions in Translation

Complex tasks like scripting are hard for both AI and humans to master, in part because of the complexity and the large variation of the input data. In our study, we found that crowd workers can extract the entities more accurately than the automated parser across a variety of paraphrases of the same query. We think this is due to the fact that the automated parser does not leverage the context when extracting, whereas humans are generally capable of understanding the context and semantic meaning in the query. This situation often occurs when certain entities are required to be understood within the context of the query. For example, one set of queries in our study asks to find some files in the "content" directory which was mis-recognized as showing the content of the files. Another example would be if the query "Copy file A to path B" was phrased instead as "Under my path B, make a copy of file A", the automated system would make the opposite prediction of which entity type is which. We propose that the crowd could assist in finding the correct order of the entities for the query using information from the appropriate man page. From the second example above, we could first pull out the description of copy (i.e., `cp`: Copy SOURCE to DEST) and ask the crowd to mark the entity order that matches the user's intent.

In addition, the queries themselves might be error-prone as they could be not formative, or informational incomplete. Neither BashOn nor Tellina offer any features that could allow users to clarify or get notified of incomplete queries (i.e., missing an entity). For example, the query "Sort my pdf files", misses a few pieces of information ("by what? under which directory?") which prevents any resources from generating the right answers. Our proposed solution to this issue is to provide similar example queries and man page templates to crowd workers. Prior work [9] has shown that providing such gold standard examples to crowd workers can increase the quality of their responses. Since the AI system has a training data set, we could find similar queries from this set, present them to crowd workers, and ask them to vote on the completion of the query. If the crowd has access to man page information, it is possible that they could identify missing entities and ask for clarification from the user. This would make the system more friendly to novice Bash users, as they currently may not recognize if something is missing from their requests.

Analysis and Challenges in Entity Extraction

Tellina's original study found that their system had some difficulty extracting certain semantic types, such as a date formatted like "the 2nd of August of 2017", or a directory named 'content', as they were using heuristics to try to cover all of these cases. However, we found that the crowd could identify these entities accurately. On the other hand, some phrases require domain knowledge to understand, and so the crowd also had some difficulties to perform the task in a different way. For example, we found that workers identified the phrase 'long-iso format' as a date, a file name, or a pattern (when it is actually not an entity), which leads to far-off outcome. Similar errors like identifying the word 'regular' from the request "List all the regular files..." as a file name, were fairly frequent. We believe this is also due to a lack of knowledge of Linux queries, despite our efforts in defining different entity types. We propose two possible ways of addressing this issue: 1) having crowd workers request clarification from the user, or 2) providing crowd workers more examples from the training data. However, we also noticed the expertise level of crowd workers varies and future work could give a higher weight to responses from workers with more expertise.

Additionally, some extracted entity lists might look correct in the voting mechanism we have currently, but they could contain small errors. Mistakes like typos, excluding the units from a size type entity, or including the word 'directory' in a file type entity are easy to interpret as correct answers. These types of misinterpretations could easily become a majority voted entity.

Analysis and Challenges in Validation

As some queries contained metadata constraints (e.g., size), we appended the command `-exec ls -lh \;` after each command before executing, which provides the time and size information of each file so that workers would be able to evaluate the outcome. This approach can be generalized in future work to cover a broader set of queries where we could allow the crowd workers to self-identify extra information they

need during the validation test and the system would adapt the commands accordingly.

One challenge that is present in our validation study is that the generated commands may modify the files, increasing the difficulty of validation. Although we only use queries that do not have the intent to modify the file system, incorrect commands may still arise, and malicious intent or bad judgment from the crowd in the entity extraction step could lead the system to generate problematic commands. One way to deal with this issue is to make copies of the file system for each command and execute the commands on the copy of the system. This preserves the original file system, but the computational cost is very high. Another solution is to identify risky commands, like `rm`, in the results before running them. This proactively prevents modification from happening, but it requires a predefined list of risky commands, which is also inefficient. We instead set the permissions of each file to read-only, preventing non-administrator users from modifying it. Future work in this domain can look into finding a way to safely execute commands in an efficient way, possibly with an interface that can generate the outcome of multiple commands executing on the same file system.

Some commands might look wrong but actually generate the desired outcome. In this case, our approach might be able to even outperform an experienced Bash command user's judgment. However, it is also possible that incorrect commands could be wrong and just happen to print out the correct answers (i.e., they have unintended side effects). Future work could look at how crowd-returned commands can be validated by the user easily from a scripting perspective.

Additionally, some commands might generate excessive information, like a long list of file names, which would be cumbersome for any human worker to validate. We considered limiting the length of the outcome, but for some queries long output could be desired. So, in order to design tasks without overloading workers with information while still being able to cover a variety of types of queries, we propose in the future dividing the outcome into pieces and distributing them to multiple workers, similarly to how we divided the file system.

The voting strategy that we used would fail to validate the results of queries that request a portion of a file system relative to other parts. For example, take the query: "Find the second largest file in my root directory". If the file system is divided into three portions where the first portion contains the largest file, second portion contains the second largest file, and the third portion contains the third largest file, none of the crowd workers could validate the result as the requested information is relative to what the rest of the system looks like.

Therefore, we propose a mechanism where we ask the crowd to report evidence they found based on their portion of the file systems that lead them to decide not to disapprove of the given outcome. This data would then be aggregated and voted on by crowd workers a final time. Using the same example above, the expected evidence from the crowd would be the number of files with size greater or equal to the return file, so there would be two files shown as evidence in the aggregation step. If the command executed returned the second one of these files, then it would be correct. Not only would this be a good method for allowing a larger range of commands to be validated, but it could also serve as a double check of the commands' validity to ensure no mistakes have been made for other types of commands where we would not normally have to aggregate evidence.

Challenges in the Workflow

Even if the all entities are correctly extracted, the correct command may still not be generated. We argue that this is an issue with the original machine learning model, as even when gold standard input data is used sometimes incorrect output is still generated. It is also possible that a modified version of our workflow could be more accurate than what we are currently using. As described previously, our workflow has a single loop in which queries are sent back to the crowd, after which commands from crowd workers and the automated system are aggregated. Changing our workflow so that it loops multiple times, or only loops when the number of incorrect commands generated is above a certain threshold are possible alternatives. Additionally, the generated commands could be ranked by different factors as well. Future work could explore the effect that variations like this have on accuracy, precision, and cost.

Using Non-expert Crowd for Data Collection

Although non-expert crowd workers are not capable of answering these queries alone to produce the training data set needed to improve an AI model, we showed that with the existing automated system they could generate valid data (i.e., query and command pairs) with certain accuracy. In the future, we plan to investigate whether having the non-expert crowd help with the program synthesis process would lower the workload from experienced script users and still generate useful training data.

CONCLUSION

In this paper, we implemented and evaluated a hybrid intelligence workflow that augments an existing program synthesis system to translate natural language query to Bash commands by leveraging non-expert crowd users. Our experimental results showed that despite not having expertise in the domain of Bash scripting, crowd workers can significantly increase the accuracy when integrated into targeted portions of this workflow. Our approach sheds light on ways that other program synthesis tools might leverage non-expert crowds to power more reliable systems.

ACKNOWLEDGEMENTS

Removed for blind review.

REFERENCES

1. 2017. ShellCheck, http://www.shellcheck.net/. (2017). http://www.shellcheck.net/ Accessed: Sep, 2017.
2. Josh Attenberg, Panagiotis G Ipeirotis, and Foster J Provost. 2011. Beat the Machine: Challenging Workers to Find the Unknown Unknowns. (2011).
3. Jeffrey P Bigham, Chandrika Jayant, Hanjie Ji, Greg Little, Andrew Miller, Robert C Miller, Robin Miller, Aubrey Tatarowicz, Brandyn White, Samual White, and others. 2010. VizWiz: nearly real-time answers to visual questions. In *Proceedings of the 23nd annual ACM symposium on User interface software and technology*. ACM, 333–342.
4. Yan Chen, Steve Oney, and Walter S Lasecki. 2016. Towards providing on-demand expert support for software developers. In *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems*. ACM, 3192–3203.
5. Justin Cheng and Michael S Bernstein. 2015. Flock: Hybrid crowd-machine learning classifiers. In *Proceedings of the 18th ACM Conference on Computer Supported Cooperative Work & Social Computing*. ACM, 600–611.
6. Robert A Cochran, Loris D'Antoni, Benjamin Livshits, David Molnar, and Margus Veanes. 2015. Program boosting: Program synthesis via crowd-sourcing. In *ACM SIGPLAN Notices*, Vol. 50. ACM, 677–688.
7. Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. 2009. Imagenet: A large-scale hierarchical image database. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*. IEEE, 248–255.
8. Aditya Desai, Sumit Gulwani, Vineet Hingorani, Nidhi Jain, Amey Karkare, Mark Marron, R Sailesh, and Subhajit Roy. 2016. Program synthesis using natural language. In *Software Engineering (ICSE), 2016 IEEE/ACM 38th International Conference on*. IEEE, 345–356.
9. Steven Dow, Anand Kulkarni, Scott Klemmer, and Björn Hartmann. 2012. Shepherding the crowd yields better work. In *Proceedings of the ACM 2012 conference on Computer Supported Cooperative Work*. ACM, 1013–1022.
10. Sumit Gulwani. 2011. Automating string processing in spreadsheets using input-output examples. In *ACM SIGPLAN Notices*, Vol. 46. ACM, 317–330.
11. Sumit Gulwani, William R Harris, and Rishabh Singh. 2012. Spreadsheet data manipulation using examples. *Commun. ACM* 55, 8 (2012), 97–105.
12. Kotaro Hara, Jin Sun, Robert Moore, David Jacobs, and Jon Froehlich. 2014. Tohme: detecting curb ramps in google street view using crowdsourcing, computer vision, and machine learning. In *Proceedings of the 27th annual ACM symposium on User interface software and technology*. ACM, 189–204.
13. Ting-Hao Kenneth Huang, Walter S Lasecki, and Jeffrey P Bigham. 2015. Guardian: A crowd-powered spoken dialog system for web apis. In *Third AAAI conference on human computation and crowdsourcing*.
14. Srinivasan Iyer, Ioannis Konstas, Alvin Cheung, Jayant Krishnamurthy, and Luke Zettlemoyer. 2017. Learning a Neural Semantic Parser from User Feedback. *arXiv preprint arXiv:1704.08760* (2017).
15. Gierad Laput, Walter S Lasecki, Jason Wiese, Robert Xiao, Jeffrey P Bigham, and Chris Harrison. 2015. Zensors: Adaptive, rapidly deployable, human-intelligent sensor feeds. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems*. ACM, 1935–1944.
16. Walter S Lasecki, Juho Kim, Nick Rafter, Onkur Sen, Jeffrey P Bigham, and Michael S Bernstein. 2015. Apparition: Crowdsourced user interfaces that come to life as you sketch them. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems*. ACM, 1925–1934.
17. Sang Won Lee and et al. 2017. SketchExpress: Remixing Animations For More Effective Crowd-Powered Prototyping Of Interactive Interfaces. In *UIST*. ACM.
18. Xi Victoria Lin, Chenglong Wang, Deric Pang, Kevin Vu, and Michael D Ernst. 2017. *Program synthesis from natural language using recurrent neural networks*. Technical Report.
19. Lena Mamykina, Bella Manoim, Manas Mittal, George Hripcsak, and Björn Hartmann. 2011. Design lessons from the fastest q&a site in the west. In *Proceedings of the SIGCHI conference on Human factors in computing systems*. ACM, 2857–2866.
20. David Merritt, Jasmine Jones, Mark S Ackerman, and Walter S Lasecki. 2017. Kurator: Using The Crowd to Help Families With Personal Curation Tasks.
21. Andrew Ng. 2016. What Artificial Intelligence Can and Can't Do Right Now. (2016). https://hbr.org/2016/11/what-artificial-intelligence-can-and-cant-do-right-now.
22. Stack Overflow. 2015. Stack Overflow, https://stackoverflow.com/. (2015). https://stackoverflow.com/ Accessed: April, 2016.
23. Man Page. 2015. Man Page, en.wikipedia.org/wiki/Manpage. (2015). en.wikipedia.org/wiki/Manpage
24. Alan Ritter, Sam Clark, Oren Etzioni, and others. 2011. Named entity recognition in tweets: an experimental study. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, 1524–1534.
25. Xin Rong, Shiyan Yan, Stephen Oney, Mira Dontcheva, and Eytan Adar. 2016. CodeMend: Assisting Interactive Programming with Bimodal Embedding. In *Proceedings of the 29th Annual Symposium on User Interface Software and Technology*. ACM, 247–258.

26. Saiganesh Swaminathan, Raymond Fok, Fanglin Chen, Ting-Hao Kenneth Huang, Irene Lin, Rohan Jadvani, Walter S Lasecki, and Jeffrey P Bigham. 2017. WearMail: On-the-Go Access to Information in Your Email with a Privacy-Preserving Human Computation Workflow. (2017).

27. Jennifer Wortman Vaughan. 2016. Making Better Use of the Crowd. (2016).