CLAI: A Platform for AI Skills on the Command Line

Mayank Agarwal , Jorge J. Barroso , Tathagata Chakraborti , Eli M. Dow , Kshitij Fadnis , Borja Godoy , Kartik Talamadupula

IBM Research

Abstract

This paper reports on the open source project – Project CLAI (**C**ommand **L**ine **AI**) – which aims to bring the power of AI to the command line interface (CLI). The platform sets up the CLI as a new environment for AI researchers to conquer by surfacing the command line as a generic environment that researchers can interface to using a simple sense-act API much like the traditional AI agent architecture. In this paper, we discuss the design and implementation of the platform in detail, through illustrative use cases of new end user interaction patterns enabled by this design, and through quantitative evaluation of the system footprint of a CLAI-enabled terminal. We also report on some early user feedback on its features from an internal survey.

1 Introduction

For decades, the AI community has pursued the vision of autonomous assistants that operate with end users inside computing systems, including "AI Softbots" [Etzioni and Lesh, 1993; Etzioni and Weld, 1994] in the 90s; and Clippy and other bots in commercial software that fell short of user expectations. A key factor behind the stagnation of progress on this vision has been that AI developers and researchers – who would be tasked with bringing AI technology to such bots – do not want to engage with the deep intricacies of the typical computing (operating) system. However, with the arrival of cloud-based ecosystems and cloud-native applications, as well as the scaleable real-world deployment of AI techniques, we are at an inflection point akin to the initial emergence of large-scale networked terminals. This is an opportune moment to transform the typical user's experience of computing systems, and imbue it with the power of AI.

One of the most common interfaces through which users (particularly developers and researchers) experience computing systems is the command line interface (CLI). Despite the many frustrations and complexities one encounters on it, the CLI remains one of the most popular computing interfaces. CLIs allow for fast invocation and execution of routine tasks; are broad and powerful in the variety of tasks that they can support; and extremely flexible in accommodating users' preferences and goals (c.f. Figure 9(h)). However, to unlock the full potential of the CLI, users must often invest a considerable amount of time into learning its various intricacies. Only after much practice, and often failure, does one gain expertise in one's CLI of choice. This substantial investment of time presents a significant barrier to entry, and is a source of great frustration for both novices and experienced CLI users (c.f. "Zombie Posts": https://meta.askubuntu.com/q/8333). Revamping the CLI experience is not a fleeting concern. In the past, we have seen integrated platforms emerge once technologies mature – this includes the Eclipse integrated development environment (IDE) from IBM, and more recently Visual Studio, Sublime, etc. However, for most applications, the rate of change is so fast that CLIs remain their primary – if not only – means of interfacing with the user. This, in addition the power of an unrestricted input interface, means that CLIs are here to stay.

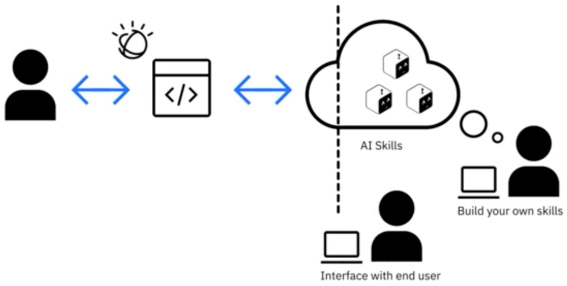


Figure 1: Project CLAI (**C**ommand **L**ine **AI**).

The CLI also presents unique challenges to AI techniques. First, the responses from AI bots or *skills* residing on the CLI must be instantaneous, and not allow for any noticeable lag. In our survey, more than 75% indicated that they expected a comeback within 3 seconds. Second, these skills cannot consume too much power (computational or otherwise), since they must ultimately run on (end) terminals; network and cloud based solutions will be restricted by the latency of the various intermediary networks.

Finally, one of the biggest issues missing from a lot of the prior work on this problem is around the broad theme of Human-AI interaction issues. That is, it is important to understand how users want to interface with the AI platform – both in terms of the interaction patterns for end-users, but also in terms of the API that developers and researchers will

use to create new skills on the platform. We focus specifically on this issue, and spend much of Section 3 and Section 4 expanding on this theme. The importance of these issues in our work is validated in our evaluation, in Section 6.

2 Related Work

For decades, the AI community has pursued assistants that operate with end users in the context of operating systems. In the early to mid 90s, researchers at the University of Washington conducted extensive academic work in this space under the umbrella of Internet Softbots [Etzioni and Lesh, 1993; Etzioni and Weld, 1994]. These were AI agents that used a UNIX shell and the web to "interact with a wide range of internet sources". These softbots provided a number of novel features, including one of the first alternative interfaces to the UNIX command line; a deliberative agent that could reason about multi-step sequences of commands; and the ability to gather information about an open world. In the late 90s, Microsoft introduced a slew of assistive agents along with their Office productivity software. Unfortunately, the (in)famous Clippy and other commercial AI sofbots fell short of user expectations. Notably, this generation of embodied assistants taught future designers valuable lessons [Cooper, 2004] for the deployment of similar agents going forward.

CLI Plugins

Recently, a number of rule-based command line assistants such as `bash-it`[1], `oh-my-zsh`[2], `thefuck`[3], `tldr`[4], etc. have emerged. These CLI assistants generally deal with correcting misspelled commands and other common errors on the command line, as well as automating commonly performed tasks using predefined scripts. While these assistants certainly make the job of working with the command line easier, they have a high maintenance burden due to the constant upkeep of the rules that form their back-end. In general, such domain specific solutions do not scale or generalize.

UbuntuWorld

In order to construct a more generalized solution to assistive CLI-centric AI, one must be able to learn the rules of the underlying system. The Linux Plan Corpus[5] is a collection of Linux sessions collected at the University of Rochester in 2003. The corpus has become a great source of data for research in this direction [Blaylock and Allen, 2004]. Our own prior work on the UbuntuWorld system [Chakraborti *et al.*, 2017] used a combination of automated planning, reinforcement learning (RL), and information retrieval to drive data-driven exploration and decision making on the CLI through a process of bootstrapping data from the AskUbuntu forum. Researchers have also attempted to use reinforcement learning (RL) for interpreting instructions in the Windows OS [Branavan *et al.*, 2009]. With recent advances – especially on sample complexity [Gu *et al.*, 2016] – learning agents are poised for a comeback in the context of operating systems.

[1] https://github.com/Bash-it/bash-it
[2] https://github.com/robbyrussell/oh-my-zsh
[3] https://github.com/nvbn/thefuck
[4] https://github.com/tldr-pages/tldr
[5] The Linux Plan Corpus: https://bit.ly/2lGzFeI

3 `CLAI`: Bringing AI to the Command Line

At the core of `CLAI` are AI plugins or "skills" that monitor every user action on the terminal, as shown in Figure 1. This is equivalent to the notion of skills in IBM's Watson Assistant (https://ibm.co/2LblJ70) or Amazon's Alexa (https://amzn.to/2ZH9Olp) – a skill is a function that can perform microtasks. Every command that the user types is piped through the backend and broadcast to all the actively registered skills associated with that user's session on the terminal. In this manner, a skill can autonomously decide to respond to any event on the terminal based on its capabilities and its confidence in its returned answer. In the rest of this section, we first outline the typical user interaction with `CLAI`; and follow that up with a list of the current interface patterns available to users and skill developers.

3.1 User Interaction with `CLAI`

A command line user has three ways of using `CLAI` skills:

`CLAI` in the Background

In this mode, the user's input most closely resembles a CLI experience prior to `CLAI`'s installation. In fact, for most commands, the user experience is entirely unchanged; e.g. user commands for which there are no associated skills.

```
>> <command>
 CLAI: augment and/or replace <command>
user: y/n/e
<stdout>
 CLAI: augment to stdout
<stderr>
 CLAI: respond to stderr
user: y/n/e
```

When a skill is invoked, `CLAI` augments its response to the `stdout` / `stderr` if and only if a high level of confidence is determined from the skill implementation backend [6]. This design choice to accommodate arbitrarily many skill plugins – each with their own confidence scoring mechanism – brings up a unique set of orchestration challenges, which can by themselves be interesting problems for ML methods. When a skill does get invoked, the user will experience one or more interface patterns enumerated here:

- `CLAI` may replace the user input command (or augment it) in order to make execution work as the user likely intended. Users see the augmented or altered input command and may approve the input variant for execution, or ask for an explanation on the command substitution rationale. The user can also hand over more power to `CLAI`, if they wish, by allowing it to execute commands without always checking back with them.
- `CLAI` may add additional information to the `stdout`.
- `CLAI` may respond to the `stderr` by (for example) providing additional information for troubleshooting purposes, or by suggesting a fix. The user, once more, is the ultimate arbiter – with the option to either act on the `CLAI`-derived suggestion, or to ask for an explanation.

[6] This is a conscious design decision in light of lessons learned from historical deployments of assistants in operating systems, often deemed to be unnecessarily obtrusive [Cooper, 2004].

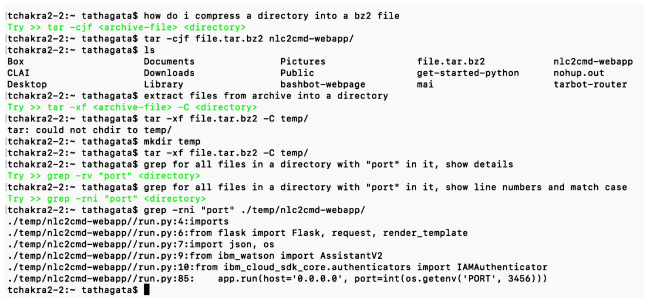


Figure 2: CLAI natural language interaction example.

CLAI Explicitly Invoked

In order to test skills or to force assistance from CLAI, a user may opt to demand a response from CLAI using the command notation below. Doing so will cause CLAI to respond with the skill that it believes to be most relevant to the context *regardless of internal confidence scoring*.

```
>> CLAI <command>
```

CLAI Forced Skill Invocation

Finally, the user can force a particular CLAI skill to respond, regardless of its confidence and those of other skills currently active, using the notation below.

```
>> CLAI <skill_name> <command>
```

3.2 CLAI Interaction Patterns

Based on these interfacing options, a few key interaction patterns emerge. While these are not intended to be exhaustive, they do capture some of the most interesting interaction types that we have explored so far with CLAI. In the following, we describe the interaction patterns and their closest match from an AI technology perspective.

Natural Language Support

This pattern allows the user to interact with the command line in natural language. For example, the user can ask `>> how do I extract file.bz2`, or tell the terminal to `>> extract all images from this archive`. An example of this pattern is shown in Figure 2.

Challenge Problem 1 The ability to turn natural language instructions into Bash commands has been a long-standing research pursuit. After all, there is a lot of data already out there in public forums and in documentation that can be readily leveraged. Particularly with recent advances in natural language processing, this problem has received renewed interest. However, most recent attempts such as Betty (https://github.com/pickhardt/betty) or the work of [Lin *et al.*, 2017; Lin *et al.*, 2018] are either heavily rule based, or do not scale beyond the examples that can be mined reliably from forums. As such, it remains an open problem today. As part of Project CLAI, we curate an open dataset in the code repository around a challenge to generate natural language interpreters of bash commands directly from man pages. We intend to host a leaderboard of competing solutions at the same location.

Figure 3: CLAI automation example.

Automation

This pattern allows the user to hand off complex tasks (achieved by one or more commands) to CLAI– e.g. deployment of an application to the cloud. Developers already write scripts to achieve some of this functionality. This feature rescues them from writing tedious automation scripts and instead provide task-dependent automation. The most obvious technology match is to established AI techniques such as automated planning and reinforcement learning. Figure 3 shows a deployed example of this pattern.

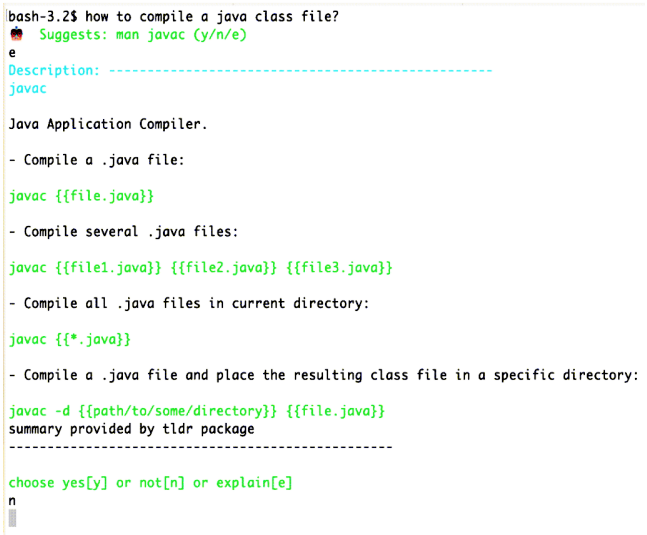


Figure 4: CLAI in-situ support & troubleshooting example, 1.

Challenge Problem 2 This brings us to the second challenge problem of learning scripts by observing users perform tasks on the command line over prolonged periods of time, either locally for the same user or from usage histories across many users. The particular use case of interest to us is the learning of recipes or pipelines for cloud deployments – we intend to host this challenge in the code repository as well.

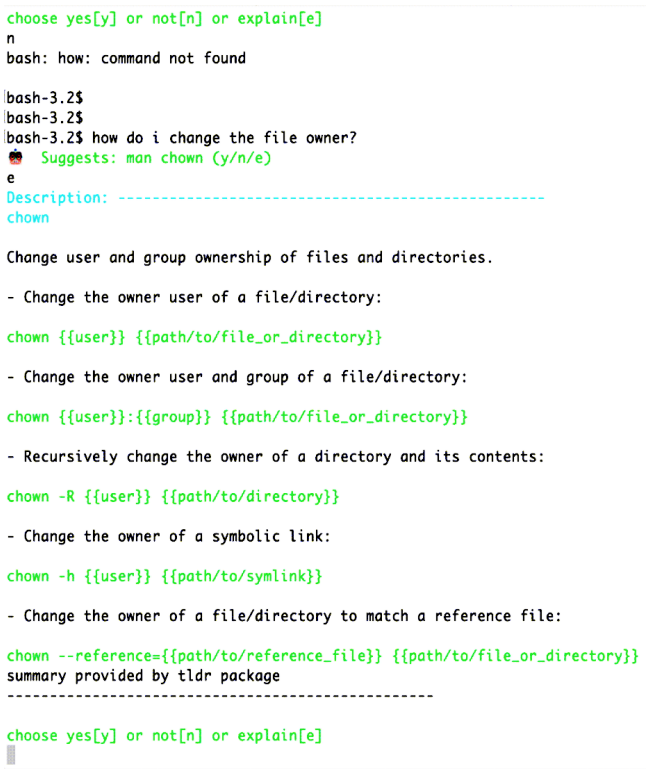


Figure 5: CLAI in-situ support & troubleshooting example, 2.

In-situ Support and Troubleshooting

Currently, when the average command line user encounters an error, the usual response is to indulge in the following loop: copy the error from the terminal, go over to a web browser, search on the internet, copy the top answer, and come back to the terminal to try it out. This is a frustrating and repetitive pattern of interaction on CLIs and integrated development environments (IDEs). The in-situ support and troubleshooting pattern of CLAI brings help from online forums and support communities directly to the terminal, so that users do not have to remove themselves from their immediate work context. This ensures that the support can be (1) *local or personalized* to the user's system; (2) *immediate*; and (3) *in-situ* without the user losing context. Figure 4 and Figure 5 show two examples of this pattern in action.

Proactive Support and Troubleshooting

In certain situations, CLAI can anticipate errors and let the user know about those errors (or even go ahead and fix them in the background) in advance. For example, it could be the case that a user might need to free up space on a cloud instance before proceeding to deploy an application. In such cases, CLAI would catch and prevent future errors that the user would otherwise encounter on the standard command line. For both troubleshooting patterns outlined here, there are obvious applications from techniques such as information retrieval [Ramos and others, 2003] and plan recognition and monitoring [Ramírez and Geffner, 2010]. Figure 6 shows an example of this pattern.

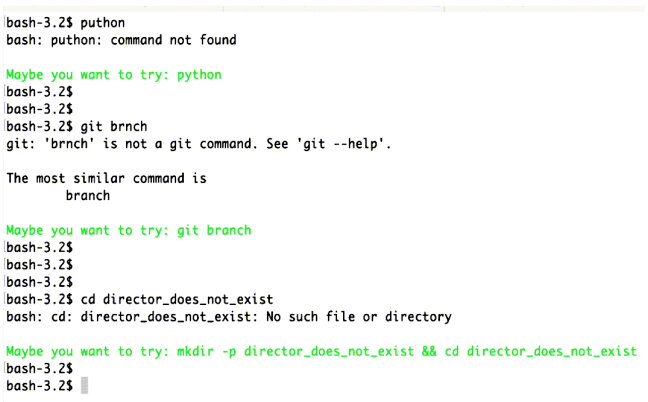


Figure 6: CLAI proactive support example.

Pedagogy

CLAI can also chime in from time to time and help the user with their proficiency on the terminal. This could involve something as simple as letting them know about new features (e.g. letting the user know that the new way of running Flask applications is >> `flask <file>` when they type in >> `python <file>`); or in the long run even retaining and guiding (for example) a new adopter of cloud platforms into becoming an expert on the cloud.

3.3 Evaluation of Interaction Patterns

Figure 7 provides feedback from our preliminary user study concerning the five interaction patterns described here. It is clear that a majority of users are in favor of all 5 patterns, and will find a use for it. The interesting thing to note is the number of `disagree` responses in the case of proactive and natural language support: we surmise that this is an artifact of these interactions being closest to human-like agency. This is an area of active future research for us.

4 CLAI for the Developer

The other important user-persona of CLAI is the developer/researcher who creates the skills. CLAI makes the Bash environment available to a skill developer via a generic "environment" API, so that the developer does not have to deal with interfacing issues and can instead focus on building their AI plugins. In order to make this very familiar to the AI community, this interface to the Bash environment allows execution of actions and sensing of the result of those actions in a manner very similar to the classic AI agent architecture [Russell and Norvig, 1995; Sutton and Barto, 1998]. This API makes the Bash environment available as another new, exciting playground for AI agents, much like OpenAI `Gym`. The CLAI API – built in `Python3` – has two major components.

The CLAI Skill API This lets a developer intercept as well as execute a callback on every user input on the command line *after the user hits return*, and lets them respond appropriately depending on what functionality their skill provides. The developers can use this to:

- Do nothing and let normal life on the command line follow. This includes doing nothing but registering an event

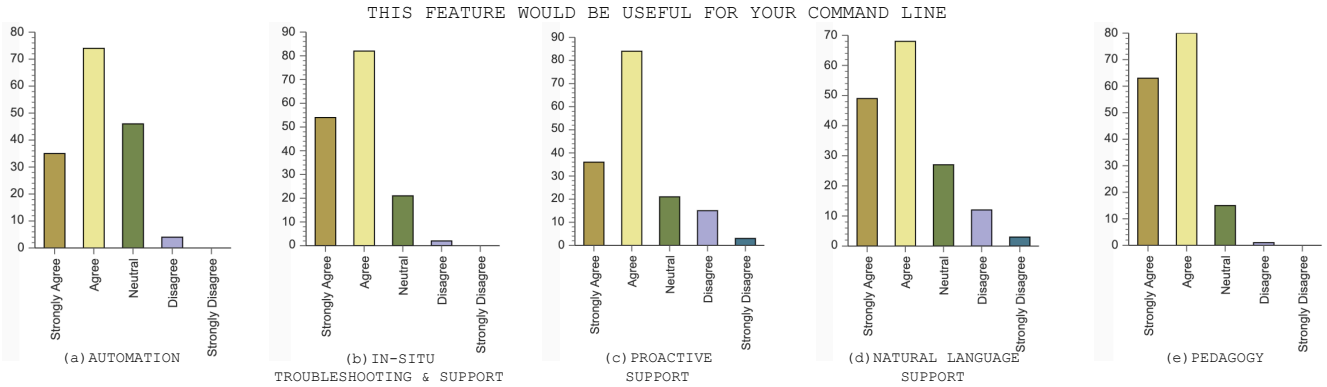


Figure 7: User results on desirability of interfacing patterns, conditioned on users who were willing to act according to CLAI's suggestions. For full results of the user study, see Figure 9 and the associated section.

to learn from that event and/or track user state.

- Add something to the user input – e.g. a flag that would make a command work.
- Replace the user input by something else – e.g. respond to a natural language request for automation.
- Respond to the outcome (e.g. error) of a user command for in-situ support and troubleshooting.
- Add something to the outcome – e.g. for proactive support and/or pedagogical purposes.

The orchestrator then decides whether to pass this on after comparing confidence values that are self-reported by the skills. The API, upon execution of this *"action"*, responds with information about the system, including state of the shell, system memory, connectivity, file system, etc. as the state information or *"percept"*. This `act-sense` cycle allows an AI agent plugged into Bash to act and learn over continued periods, either by itself [Chakraborti *et al.*, 2017] or in the course of prolonged interaction with the user.

The CLAI Orchestration API The orchestration layer comes with another unique set of challenges. In general, there are at least two orchestration methods: *apriori*, where the orchestrator acts as a filter and decides which plugin to invoke based on the input; and *Posterior*, where all plugins listen and respond, and let the orchestrator pick the best response (this is the current setup with a selector that picks the skill with the highest confidence). The apriori option is likely to have a significantly smaller system footprint, but involves a single bottleneck based on the accuracy of the classifier which determines which plugin to invoke. Furthermore, this requires that the orchestrator design be cognizant of the list of plugins and their capbilities – this is unrealistic. The *posterior* option – despite increased latency and computational load – keeps the skill design process independent from the orchestration layer *as long as the confidences are well calibrated*. This can be achieved by learning from user responses to CLAI actions, either directly from their y/n/e responses or indirectly by observing what command they executed after a suggestion, and gradually adapting a normalizer over the confidences self-reported by the skills. The ability to learn such patterns can also be used to eventually realize a healthy mix of apriori and posterior orchestration strategies [Upadhyay *et al.*, 2019].

4.1 Packaged Skills

We now describe the skills available in CLAI by default. Full details of these skills are available in the documentation.

nlc2cmd This is the canonical example of a natural language interface to the terminal. It connects to a Watson Assistant instance to interpret user intents, and translate those to the appropriate `tar` and `grep` commands.

CLAI **fixit** This skill provides help in response to the last command executed, by echoing back the response from the `thefuck` plugin. This agent is meant to demonstrate how to build an existing Bash plugin into the CLAI platform.

Man Page Explorer This agent interprets questions posted in natural language, and responds with the most relevant command it can find from the manual (man) pages already installed in the system. It also augments its response with a concise description of the man page of the suggested command using the `tldr` plugin. This is an illustration of both natural language support as well as plugin integration.

CLAI **help!** This fires whenever there is an error, and fetches the most relevant post it can find on Unix Stack Exchange using an Elasticsearch index.

CLAI **howdoi** This is similar to the retrieval agent, but is invoked via explicit user input instead of an error. It lets the user ask any question in natural language and responds with the most relevant answer from Unix Stack Exchange.

Kube Bot This is a stateful agent that can automate tasks involving Docker [Merkel, 2014] and Kubernetes [Brewer, 2015] that require execution of a sequence of actions by harnessing the power of automated planning, as an instance of the automation use case. The role of the automated planner here is to generate scripts that would otherwise have to be specified manually. In addition to automating the lengthy deployment pipeline, the `YAML` file that currently needs to be written manually is generated automatically by the Kube Agent by: 1) monitoring user activities on the terminal; 2) pinging the cloud account for the types of services available;

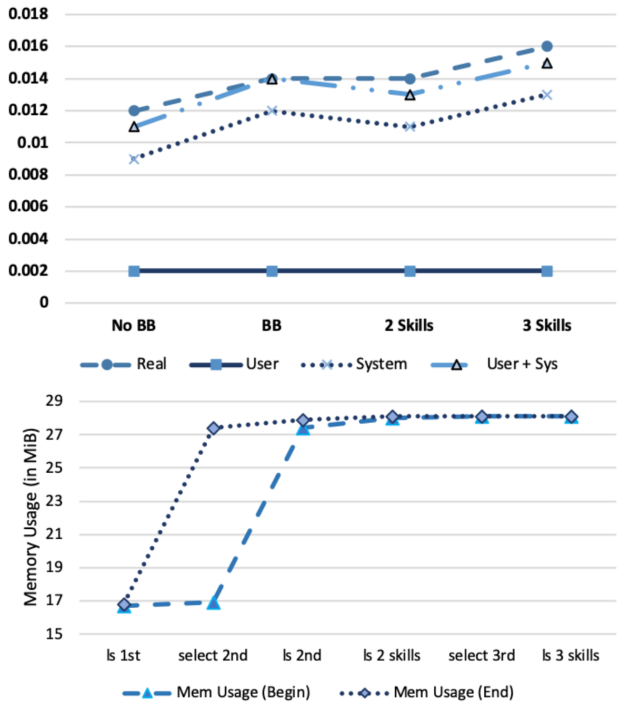


Figure 8: CLAI's footprint: (a) Latency; (b) Memory Usage.

and 3) parsing the Dockerfile. This skill saves developers a large amount of routine effort, and was developed specifically keeping the transition to cloud-based systems in mind.

4.2 Pending Features

The following discussion highlights three features that are not part of the first roll out of the CLAI platform, but that have heavily featured as desired additions to the CLAI API in the course of feedback from potential skill developers.

Simulator Access to a Bash simulator allows the developer of a plugin to act and learn by itself either *offline* – e.g. to train an RL agent – or *online* – e.g. to estimate the effects of the user's actions or its own suggestions in real time. The former – in the mould of [Chakraborti *et al.*, 2017] – is in the process of being integrated into the CLAI platform in the form of a containerized environment that can load and save histories of user activities from the host machine. The latter is an intriguing possibility, though most likely out of scope at the moment due to the time and memory requirements associated with regular Bash interactions (since it will require spawning off a copy of the file system in real time).

Auto-complete Currently, all the processing in CLAI occurs only after the user hits enter. This means features like auto-complete are currently out of scope. This can be a powerful feature especially on enterprise cloud platforms, since auto-complete features can be built off of data that is particular to those platforms – thus providing unique value to clients.

5 Internal System Evaluation

One of the primary challenges in deploying a framework like CLAI is ensuring that the resources consumed do not hinder the user experience on the command line. This fact is made evident from our survey of end users (which we report on fully in the next section) – a whopping 80% of the respondents (and 93% of developers/devops) required a latency of less than 3 seconds, with more than half that number requiring an even more stringent sub-second latency (see Figure 9(g) for details). It is thus clear that in addition to the features offered, the *footprint* of the CLAI system is important. We provide some numbers from internal benchmarking with respect to CLAI's latency (time to return) and memory consumption.

System Latency We report on two cases here: first, we look at the overall, end-to-end latency of the system as it pertains to returning a response to the user on a client terminal. Figure 8(a) showcases the latency under various conditions – the first three lines indicate the standard way of measuring time on the CLI, while the fourth is sometimes a more accurate representation of compute time than the real (wall-clock) time elapsed. As seen, there is a slight uptick in the processing time between the absence and presence of CLAI, which is to be expected. The second case we look at is much more revealing: this is the scaling of CLAI in terms of the number of skills that are available. The most interesting point to note is that there is absolutely no change in the user (client) time trend, while the other times scale up. This is encouraging, since it suggests that the client side – which cannot be massively scaled up from a compute perspective, unlike the server side which is often cloud-deployed – shows near-constant latency. In all cases, our latency is a fraction of the tolerance limit suggested by our user study.

Memory & Processing We also measure CLAI's footprint in terms of memory consumption. We show the scaling of the system's memory usage with skill addition in Figure 8(b): the various events are the adding of skills (confirmed via the `ls` command) and the selection of the added skill. The key takeaway from this graph is that adding skills and invoking them does not increase the memory footprint of the system in any significant way. This should also be readily evident from the videos of different skills in action (provided at the end).

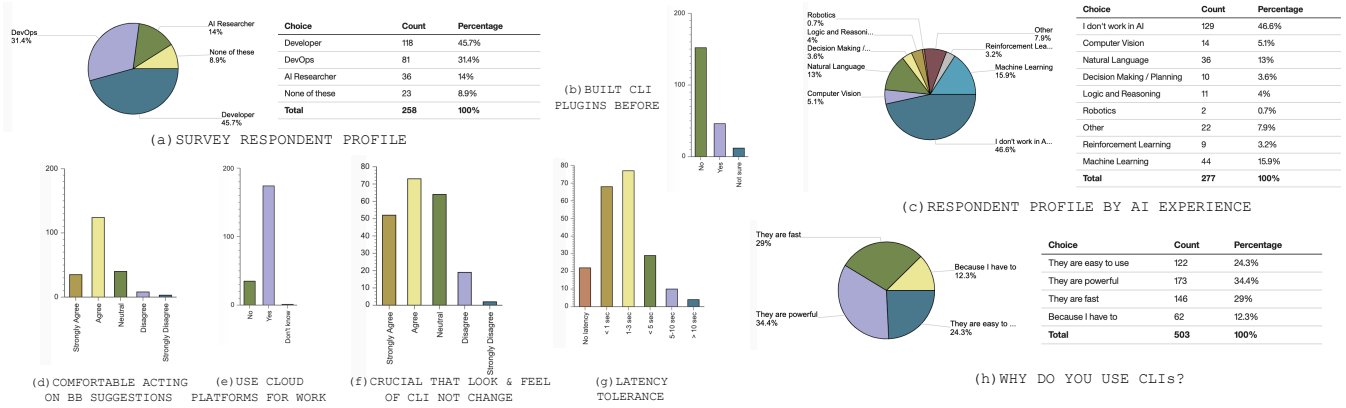
6 Preliminary User Study

We now report on the user feedback on the CLAI system from an internal survey. There were 235 responses.

6.1 Aggregate Results

Figure 9(a) profiles the survey respondents in aggregate. More than three-quarters of the responses came from respondents who identified as either developers or devops, while only around 14% of the respondents identified as AI practitioners. This indicates the potential for CLAI to positively impact communities that have hitherto not had too much interaction with state-of-the-art AI techniques and technologies.

AI Experience Figure 9(c) zooms in on the respondents' AI interests. Roughly over half of the respondents (belonging to all categories among developer, devops, and researcher)

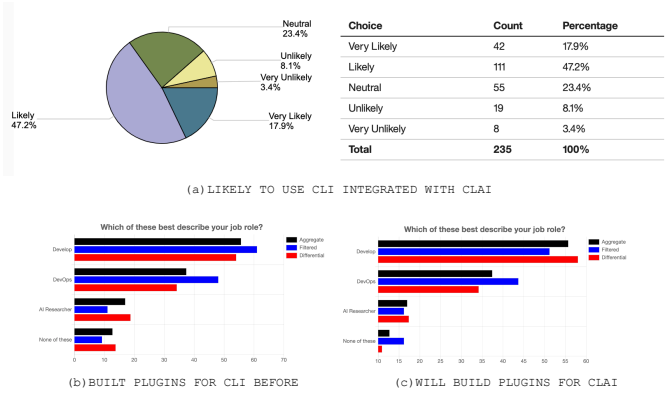


Choice	Count	Percentage
Developer	118	45.7%
DevOps	81	31.4%
AI Researcher	36	14%
None of these	23	8.9%
Total	**258**	**100%**

Choice	Count	Percentage
I don't work in AI	129	46.6%
Computer Vision	14	5.1%
Natural Language	36	13%
Decision Making / Planning	10	3.6%
Logic and Reasoning	11	4%
Robotics	2	0.7%
Other	22	7.9%
Reinforcement Learning	9	3.2%
Machine Learning	44	15.9%
Total	**277**	**100%**

Choice	Count	Percentage
They are easy to use	122	24.3%
They are powerful	173	34.4%
They are fast	146	29%
Because I have to	62	12.3%
Total	**503**	**100%**

Figure 9: Aggregate results from the preliminary user study.



Choice	Count	Percentage
Very Likely	42	17.9%
Likely	111	47.2%
Neutral	55	23.4%
Unlikely	19	8.1%
Very Unlikely	8	3.4%
Total	**235**	**100%**

Figure 10: User study results on the desirability of using the CLAI platform, and developing skills for it.

had some past AI experience. Of these, Machine Learning was the top area, with Natural Language (Processing) a close second. These results are expected, and follow the general trend in the AI world today, particularly when it comes to the non-research population. However, it is also interesting to note that there is still a reasonable population of respondents who are familiar with *classical* AI disciplines such as Decision Making, Logic & Reasoning, and Reinforcement Learning. This wide distribution across AI topics (including Computer Vision) reinforces the notion that CLAI is of use to real-world AI practitioners across the spectrum of AI areas.

Journey to Cloud One of the unique value propositions of CLAI is to offer instantaneous and on-premise support for new adopters of cloud platforms (c.f. **Challenge 2**). This is reflected in Figure 9(e), where nearly 4 out of 5 respondents report using cloud platforms for their work.

6.2 Adoption Tolerance

As with the introduction of any new technology, we measure and report the tolerance and appetite of the end-user for the tool being offered. This was done via a variety of questions, most specifically represented in Figure 9(f) and Figure 9(g). The former talks to the tendency of developers and other power-users to not want overt changes to the CLI that they know and love; indeed, a majority of respondents did not want changes in the appearance and handling of their CLI. The latter – Figure 9(g) – talks to users' patience with processing time and latency in general. Unsurprisingly, users are not willing to tolerate latencies of more than 3 seconds; however, there is a sizeable contingent of respondents who are happy to trade-off a bit of latency for the added AI boost.

6.3 Using & Extending CLAI

Finally, we also surveyed users on whether they would use a version of the CLI integrated with CLAI – these results are aggregated in Figure 10(a). The vast majority of users report that they are likely to use CLAI. This provides an evaluation for our description of the user interaction patterns in Section 3. Furthermore, in order to evaluate our contribution in Section 4, we present Figure 10(b) and Figure 10(c), which respectively show the breakdown by role of respondents who have built plugins for the CLI *previously*, and those who would build *new* AI-based plugins for CLAI. It is particularly informative to note the difference in numbers between the `developer` and `researcher` job roles.

7 Try it out!

This concludes a detailed overview of Project CLAI. It poses an exciting new environment for AI researchers to conquer, with its own unique challenges that require skills to respond near instantaneously with little system footprint, while navigating human-computer interactions issues over time. We have attached a few links below that pertain to this project.

- Project CLAI Overview: http://ibm.biz/clai-video
- Interactive Survey: You can use this service to provide feedback on CLAI, and eventually to explore differentiated results across different participant subgroups: http://ibm.biz/clai-survey
- Open source repo: http://ibm.biz/clai-home

Acknowledgements
We would like to express our heartfelt gratitude to all those who contributed in bringing this effort to fruition, particularly

Yasaman Khazaeni for lending her experiences from developing the orchestration layer in Watson Assistant *Alpha*, as well as for her endless support for the project. We would also like to thank Talia Gershon, Kara Kotwas, Caitlin Gettinger, Amanda Shearon, John Bister, Vittorio Caggiano, and the rest of the IBM ETX (Emerging Technology Experiences) Team for helping us with the survey and providing project guidance.

References

[Blaylock and Allen, 2004] Nate Blaylock and James F Allen. Statistical Goal Parameter Recognition. In *ICAPS*, 2004.

[Branavan *et al.*, 2009] Satchuthananthavale RK Branavan, Harr Chen, Luke S Zettlemoyer, and Regina Barzilay. Reinforcement Learning for Mapping Instructions to Actions. In *ACL/AFNLP*, 2009.

[Brewer, 2015] Eric A Brewer. Kubernetes and the path to cloud native. In *Proceedings of the Sixth ACM Symposium on Cloud Computing*, pages 167–167, 2015.

[Chakraborti *et al.*, 2017] Tathagata Chakraborti, Kartik Talamadupula, Kshitij P Fadnis, Murray Campbell, and Subbarao Kambhampati. UbuntuWorld 1.0 LTS – A Platform for Automated Problem Solving & Troubleshooting in the Ubuntu OS. In *IAAI/AAAI*, 2017.

[Cooper, 2004] Alan Cooper. *The Inmates are Running the Asylum: Why High-tech Products Drive Us Crazy and How to Restore the Sanity*. Sams Indianapolis, 2004.

[Etzioni and Lesh, 1993] Oren Etzioni and Neal Lesh. Planning with Incomplete Information in the UNIX Domain. In *Working Notes of the AAAI Spring Symposium: Foundations of Automatic Planning: The Classical Approach and Beyond*, pages 24–28, 1993.

[Etzioni and Weld, 1994] Oren Etzioni and Daniel Weld. A Softbot-Based Interface to the Internet. *Communications of the ACM*, 37(7):72–76, 1994.

[Gu *et al.*, 2016] Shixiang Gu, Timothy Lillicrap, Ilya Sutskever, and Sergey Levine. Continuous deep q-learning with model-based acceleration. In *International Conference on Machine Learning*, pages 2829–2838, 2016.

[Lin *et al.*, 2017] Xi Victoria Lin, Chenglong Wang, Deric Pang, Kevin Vu, and Michael D Ernst. Program synthesis from natural language using recurrent neural networks. *University of Washington Department of Computer Science and Engineering, Seattle, WA, USA, Tech. Rep. UW-CSE-17-03-01*, 2017.

[Lin *et al.*, 2018] Xi Victoria Lin, Chenglong Wang, Luke Zettlemoyer, and Michael D Ernst. Nl2bash: A corpus and semantic parser for natural language interface to the linux operating system. *arXiv preprint arXiv:1802.08979*, 2018.

[Merkel, 2014] Dirk Merkel. Docker: lightweight linux containers for consistent development and deployment. *Linux journal*, 2014(239):2, 2014.

[Ramírez and Geffner, 2010] Miguel Ramírez and Hector Geffner. Probabilistic plan recognition using off-the-shelf classical planners. In *Twenty-Fourth AAAI Conference on Artificial Intelligence*, 2010.

[Ramos and others, 2003] Juan Ramos et al. Using tf-idf to determine word relevance in document queries. In *Proceedings of the first instructional conference on machine learning*, volume 242, pages 133–142. Piscataway, NJ, 2003.

[Russell and Norvig, 1995] Stuart J Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Pearson Education Limited, 1995.

[Sutton and Barto, 1998] Richard S Sutton and Andrew G Barto. *Introduction to Reinforcement Learning*. MIT Press Cambridge, 1998.

[Upadhyay *et al.*, 2019] Sohini Upadhyay, Mayank Agarwal, Djallel Bounneffouf, and Yasaman Khazaeni. A Bandit Approach to Posterior Dialog Orchestration Under a Budget. *NeurIPS Conversational AI Workshop*, 2019.