
World of Bits: An Open-Domain Platform for Web-Based Agents

Tianlin (Tim) Shi^{1,2} Andrej Karpathy² Linxi (Jim) Fan¹ Jonathan Hernandez² Percy Liang¹

Abstract

While simulated game environments have greatly accelerated research in reinforcement learning, existing environments lack the open-domain realism of tasks in computer vision or natural language processing, which operate on artifacts created by humans in natural, organic settings. To foster reinforcement learning research in such settings, we introduce the World of Bits (WoB), a platform in which agents complete tasks on the Internet by performing low-level keyboard and mouse actions. The two main challenges are: (i) to curate a diverse set of natural web-based tasks, and (ii) to ensure that these tasks have a well-defined reward structure and are reproducible despite the transience of the web. To tackle this, we develop a methodology in which crowdworkers create tasks defined by natural language questions and provide demonstrations of how to answer the question on real websites using keyboard and mouse; HTTP traffic is cached to create a reproducible offline approximation of the website. Finally, we show that agents trained via behavioral cloning and reinforcement learning can complete a range of web-based tasks.

1. Introduction

Over the last few years, we have witnessed significant progress in developing agents that can interact with increasingly complex environments (Mnih et al., 2015; Silver et al., 2016; Levine et al., 2016). Critical to this progress are not only the core learning algorithms (Sutton et al., 1999; Mnih et al., 2015; Schulman et al., 2015a) and the associated techniques for learning at scale (Mnih et al., 2016), but simulated environments that feature complex dynamics and help benchmark our progress (e.g., Belle-mare et al. (2013); Mikolov et al. (2015); Todorov et al.

¹Stanford University, Stanford, USA ²OpenAI, San Francisco, USA. Correspondence to: Tianlin (Tim) Shi <tianlin@cs.stanford.edu>.

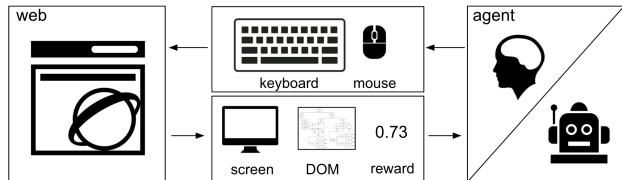


Figure 1. Agents in the World of Bits perceive the screen pixels, the DOM (with element coordinates grounded in the image), and a reward, and output keyboard and mouse commands.

(2012); Johansson et al. (2016)). However, simulated environments are intrinsically limited: agents in such environments never experience the sheer breadth of experience of the real world, and thus they miss out on important semantic knowledge crucial for developing intelligence. For control tasks, it is possible to work with realistic environments in robotics, but the complexity of physical hardware constraints efficient data gathering and rapid iteration. Even for narrow domains such as grasping (Levine et al., 2016; Pinto & Gupta, 2016), the cost and effort of large-scale data collection is daunting.

To address this, we introduce *World of Bits (WoB)*,¹ a learning platform that uses the web as a rich source of open-domain environments. In WoB, an agent receives its observations in the form of the Document Object Model (DOM) of a webpage and its rendered pixels, and accomplishes *web tasks* by sending mouse and keyboard actions. The use of web as a learning platform offers three benefits:

Open-domain. By allowing agents to interact with the web, we open up the world’s supply of websites as a rich source of learning environments and application domains. Since agents directly work with the UI, we can use existing web infrastructure without designing specialized APIs.

Open-source. Unlike robotics, WoB is digital, which enables fast iteration and massive scaling. Webpages are open-source and consist entirely of HTML/CSS/Javascript, which is easy to inspect and change dynamically.

Easy to collect data. Because agents use same interface as humans do, it is possible to crowdsource human demonstrations of a web task from anyone with an access to a web browser, keyboard and mouse at a low cost. This unlocks

¹in contrast to the *world of atoms* <https://goo.gl/JdLQGT>

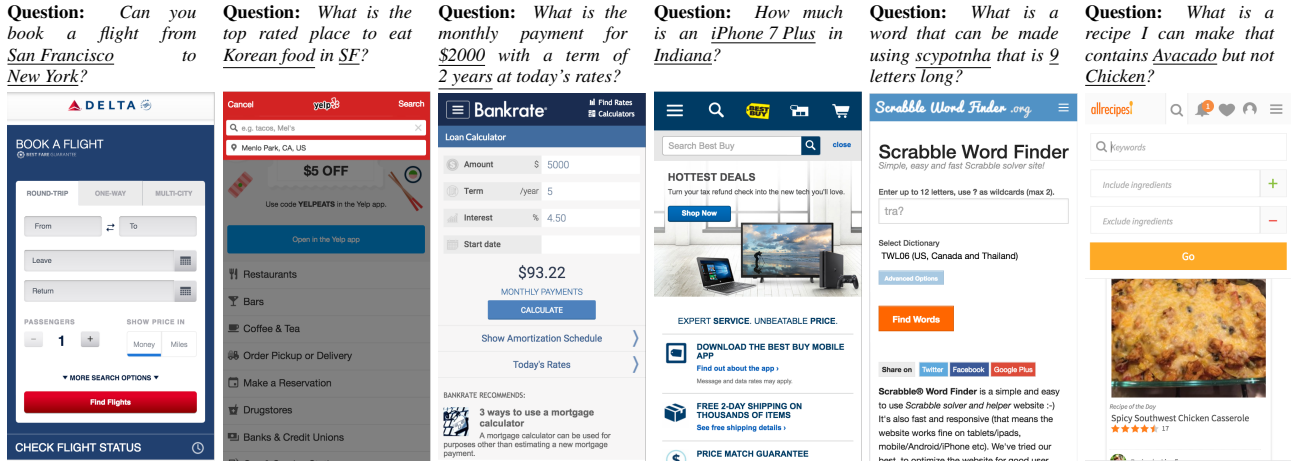


Figure 2. Examples of open-domain web tasks. Each web task is specified by a natural language query template and slot values (underlined), and the agent receives positive reward for clicking on the correct answer. We crowdsource the creation of these tasks.

the potential for large-scale data collection.

While WoB specifies a platform, the main conceptual challenge is to define meaningful web tasks in a scalable way. In Section 2.2, we start by constructing the Mini World of Bits (MiniWoB), 100 web tasks (see Figure 7 for examples) of varying difficulty, in which the reward function is manually constructed.

Next, in Section 2.3, we describe FormWoB, which consists of four web tasks based on real flight booking websites. The main difficulty here is that websites are constantly changing, and yet we would like to package them into reproducible research environments for our agents. To this end, we use a man-in-the-middle proxy to capture and replay live HTTP traffic, building up an approximation of the live website.

Finally, inspired by large datasets such as ImageNet in computer vision (Deng et al., 2009) and SQuAD in NLP (Rajpurkar et al., 2016), we would like to scale up to a diverse set of web tasks without manual effort on each web task. To tackle this, we develop a methodology based on crowdsourcing that effectively casts web tasks as question answering (Section 2.4). First, we ask crowdworkers to write queries that can be answered by interacting with a given website. Each query is defined by a *query template* and *slot values* (e.g., “New York”) that fill the template slots (See Figure 2 for examples). Positive reward is given if an agent clicks on the correct answer. We create a dataset, QAWoB, which has 11,650 queries (from 521 templates). We collected initial demonstrations for four of the templates, with one demonstration per query. Collecting demonstration for the full dataset is on-going work.

To benchmark a standard approach, we evaluate the performance of convolutional neural networks that take as input the image and text from the DOM and outputs keyboard

and mouse actions. We train these models using both supervised learning and reinforcement learning, and show that in some cases we can generalize across different queries of the same template. However, our overall error rates remain relatively high, suggesting that the proposed benchmark leaves a lot of room for improvement.

2. Constructing Web Tasks

In this section, we describe a progression of three techniques for creating web tasks, MiniWoB (Section 2.2), FormWoB (Section 2.3), and QAWoB (Section 2.4).

2.1. Web as an Environment

To interact with a web browser, we developed our platform on top of OpenAI Universe (<http://universe.openai.com/>), which allows one to package nearly arbitrary programs into Gym (Brockman et al., 2016) environments suitable for reinforcement learning. Specifically, we package a Chrome browser inside a Docker container, which exposes a Gym interface for the agent to interact with. At each time step t , the agent receives an observation, which consists of the raw screen pixels $\mathcal{I} \in \mathbb{R}^{W \times H \times 3}$ (e.g. of resolution $1024 \times 768 \times 3$), the text DOM \mathcal{D} , and a scalar reward signal r . Each element of \mathcal{D} is localized in the image by a 4-tuple (x, y, w, h) , denoting its bounding box. The agent communicates back a list of actions, which can be 1) a KeyEvent (e.g. hold down the `k` button), or 2) a PointerEvent (e.g. move the mouse to location (140, 56) while holding down the left mouse button). Then the agent obtains reward r_t which is defined by the specific web task.

2.2. Minimalistic Web Tasks: MiniWoB

Inspired by the ATARI Learning Environment (Bellemare et al., 2013), we designed a benchmark of 100 reinforce-

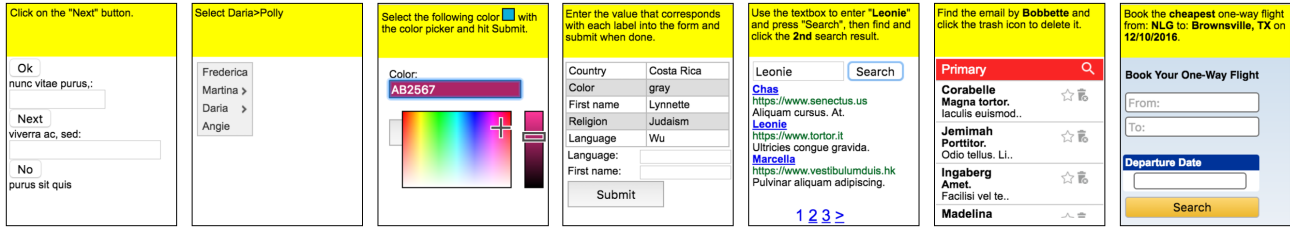


Figure 3. 7 of the 100 MiniWoB web tasks, ranging from simple (left) to more complex (right).

ment learning environments called Mini World of Bits (MiniWoB) that share many of the characteristics of live web tasks (interacting with buttons, text fields, sliders, date pickers, etc.) and allows us to study these challenges in a controlled context. Since the web offers powerful visual design tools, the average MiniWoB environment is only 112 lines of HTML/CSS/JavaScript. Each MiniWoB environment is an HTML page that is 210 pixels high, 160 pixels wide (i.e. identical to the ATARI environment dimensions) — the top 50 pixels (in yellow background) contain the natural language task description (randomly generated) and the 160×160 area below is for interactions. The rewards range from -1.0 (failure) to 1.0 (success) and are weighted linearly with time to encourage fast completion time. See Figure 7 for examples.

2.3. Live Web Tasks: FormWoB

While it is possible to create web tasks from scratch (e.g. MiniWoB), the Internet already offers a massive repository of websites. In this section we describe an approach that allows us to convert these websites into web tasks.

Since websites change over time and since we do not wish to spam websites with requests while the agent is training, we need to create an offline *approximation* that the agent can interact with. To do this, when we collect human demonstrations, we use a proxy to record all HTTP requests and responses between the agent and the website. To train and evaluate agents on a web task, we use the proxy to handle all requests with the recorded responses.

We also use requests to define reward functions. Form-filling tasks involve making a final request to the website with a set of key-value pairs (e.g., {from: DEN, to: JFK}). We define the reward function as the fraction of key-value pairs that match those in human demonstrations.²

When an agent performs an action that generates a request never seen during human demonstrations (i.e., a cache miss), we immediately end the episode with zero reward. This provides a lower bound on the true reward if the agent

²Ideally, we would require exact match, but this resulted in too sparse of a reward signal to train and evaluate with.

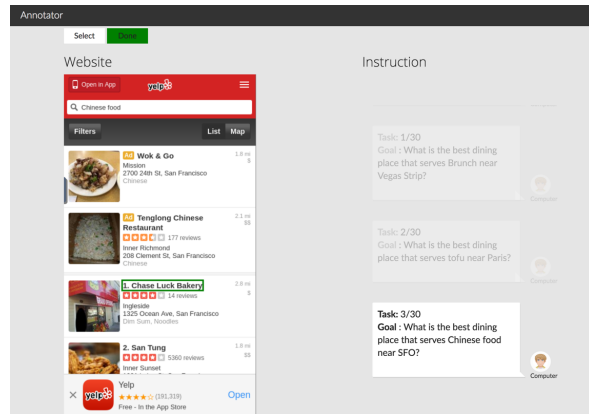


Figure 4. Our crowdsourcing interface for collecting human demonstrations on the web. The left side streams visual observations using VNC and the right side displays queries. All observations and actions are recorded. At the end of episode, the worker marks a DOM element as the answer (green box).

were to interact with the real website (assuming all rewards are non-negative), since all action sequences that result in a cache miss receive the minimum possible reward.

FormWoB benchmark. We applied this approach to four flight booking websites (United, Alaska, AA, and JetBlue). On each website, an agent must fill out a form and click on the submit button. The form filling process requires a diverse set of interaction skills, such as typing cities in a text box using autocomplete, using a date picker, etc. For each website, there is a query template parameterized by the following fields: an origin airport, a destination airport, a departure date, and a return date. Airport names are sampled from 11 major US cities, and dates are sampled from March 2017. We created 100 different instantiations for each query template, and collected on average 1 episode of human demonstration for every query.

2.4. Crowdsourcing Web Tasks at Scale: QAWoB

To take full advantage of the scale and diversity of the web, we now present a more scalable approach to generating web tasks that does not involve specifying the reward functions manually for each web task. The key is cast web tasks as question answering, and solicit questions from crowd-

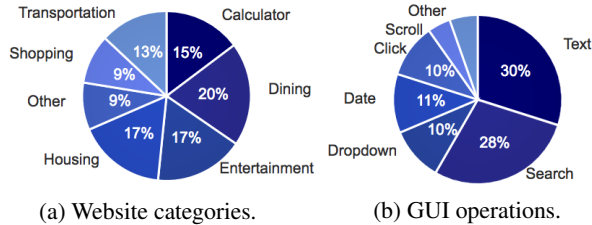


Figure 5. QAWoB contains queries from a diverse set of categories and require many types of operations.

workers. The approach has two stages:

Stage 1. A worker provides a website (e.g., `yelp.com`) and a query template (e.g., “What is the cheapest restaurant that serves (type of food) near (geographic location)?”). We also ask workers to generate multiple slot values for each template (e.g. “brunch” / “San Francisco”, “hamburger” / “JFK international airport”, etc.).

Stage 2. Next, a worker takes a query from stage 1 and uses our demonstration interface to answer it (see Figure 4).³ The interface has a “Select” button, which allows the worker to mark the DOM element of the webpage corresponding to the answer. We define the (very sparse!) reward for the task to be 1 only if an agent clicks on the annotated DOM element.

We encouraged workers to be creative when they pick the website and the queries so that we can capture a wide distribution of online activities. However, we do impose some constraints. For instance, in the instruction we discourage queries that require too much reading comprehension (e.g. “How many royal families are mentioned in Game of Thrones?” on `wikipedia.org`). We also require that the website be mobile-friendly, because the learning environment operates in mobile view.

QAWoB benchmark. Our crowdsourced QAWoB dataset has 521 query templates. The majority of the templates have 2 slots, while the average is 2.54. We gather 10 - 100 slot values per template, resulting in 13,550 total queries. 11,650 of the queries have corresponding answers. In most cases, one needs to navigate through multiple screens or menus, and perform a search before locating the answer. This makes the problem particularly hard for pure RL approaches, as random exploration has little chance to stumble upon the goal state.

We label 100 of the templates with the sequence of GUI operations required to find the answer. Note that there are multiple ways to accomplish the task and some of the operations can be reordered, so we only provide one of the shortest paths. There are 7 GUI operations: search, text

³ The interface runs VNC connected to a WoB docker container running a browser.

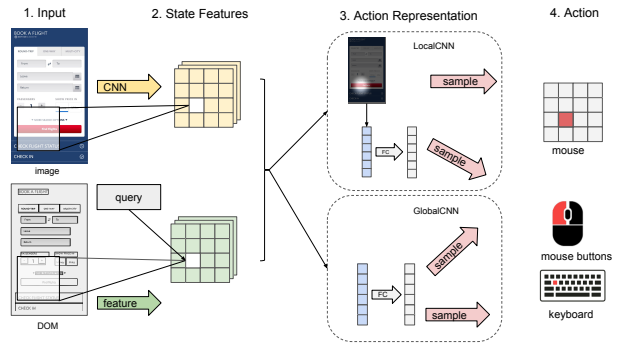


Figure 6. CNN architecture. **GlobalCNN:** The CNN representation is concatenated with DOM features into a global feature vector, which is used to predict mouse and keyboard events. **LocalCNN** (depicted): The mouse position defines an attention mechanism that pools the CNN features.

(any textbox that is not a search box), date, dropdown, scroll, click (any click that is not part of the other operations), and other (less common GUI widgets like sliders). We also organize the templates into 7 categories: dining, entertainment, housing, transportation, shopping, calculator, and other. Figure 5 shows the distribution of categories and GUI operations.

3. Training Web Agents

To build an agent for the WoB setting requires modeling a novel state space (images and DOM) and action space (keyboard and mouse).

3.1. Model

State space. The state consists of a color image I , the DOM \mathcal{D} , and the query q . The color image I has size $W \times H \times 3$. The DOM is a list of text elements, with bounding boxes (x, y, w, h) to represent their spatial relations. For MiniWoB, the query is natural language. For FormWoB and QAWoB, we assume a semantic frame is extracted for q , in the format of (template, slots).

Action space. We model the cursor position $m = (m_x, m_y) \in [0, W) \times [0, H)$ with a multinomial distribution over the positions in a regular grid over the image.⁴ We model the mouse actions with a multinomial distribution over four possibilities: `no-op`, `click`, `drag`, `scroll-up`, `scroll-down`. Finally, the key actions also follow the multinomial distribution. We found that giving the agent unrestricted access to the keyboard is impractical, as the agent may press key combinations such

⁴We also experimented with a Gaussian distribution but found it inadequate due to its unimodal shape.

as ‘CTRL+w’, which closes the window. Therefore, in addition to keys we create atomic actions out of common and safe key combinations, such as ‘CTRL+c’ (copy), ‘CTRL+v’ (paste), and ‘CTRL+a’ (select all).

Architecture. Our model (see Figure 6) first processes the image using a Convolutional Neural Network (CNN). For DOM, we compute a text feature map based on the matching between query and DOM. Then the two maps are concatenated into a joint representation. On top of this we develop two variants: first we flatten the features and feed them directly through a fully-connected layer (GlobalCNN). Since we had the intuition that local features alone should suffice to characterize the action, we also examine a LocalCNN architecture to capture the intuition that agent should attend to where cursor is. So the mouse distribution is used as soft attention (Bahdanau et al., 2014) to average the CNN features into a global representation to predict mouse buttons and keyboard events.

3.2. Optimization

We train models on web tasks by sequencing behavior cloning and reinforcement learning.

Behavior cloning. Since our web tasks can have very long time horizons and sparse rewards, a naive application of reinforcement learning will likely fail. Therefore, we pretrain our networks by optimizing a supervised learning objective (Pomerleau, 1989) on demonstrations (which were used to define the reward in the first place). Since a typical recording might have thousands of frames, we filter out frames where there was no action to obtain a dataset of state-action tuples.

Reinforcement learning. Policies trained with supervised learning suffer from compounding errors, so we fine tune the policies by optimizing the expected reward using a policy gradient method (Sutton et al., 1999). In particular, we use the Asynchronous Advantage Actor-Critic (A3C) (Mnih et al., 2016) and estimate the advantage using the Generalized Advantage Estimation (Schulman et al., 2015b) with the standard settings $\gamma = 0.9$, $\lambda = 0.95$.

4. Experiments

Our goal in this section is to establish baselines that current techniques provide on web environments, and highlight the challenges for future work in this area.

4.1. Results on Synthetic Web Tasks (MiniWoB)

Demonstration data. We collected 10 minutes of human demonstrations on each of the 100 MiniWoB environments (about 17 hours total). Unlike the FormWoB and QAWoB settings, the MiniWoB dataset contains interactions that re-

quire dragging and hovering (e.g. to trigger a menu expansion). Therefore, we process the demonstrations at regular 83 millisecond intervals (12 frames per second) to extract approximately 720,000 state-action pairs. With grid-points spaced 8 pixels across the 160 pixel area, we obtain a 20×20 grid and 3 possible actions (move, drag, click), leading to a total of $20 \times 20 \times 3 = 1200$ possible actions.

Model. In these experiments we use a 6-layer feedforward network that takes the $210 \times 160 \times 3$ image, and applies 5 convolutional layers with 5×5 filters of stride 2 and sizes [16, 24, 32, 48, 32]. We then average pool the representation and pass it through one fully-connected layer of 384 units and another to compute the logits for the mouse and key actions. Surprisingly, we found that feeding in the previously taken actions hurts performance because the agent learns to use continuous paths similar to humans and develops a tendency to meander, which negatively impacts exploration in many environments.

Evaluation. For the purposes of evaluation, a robust statistic to use is the *success rate* (SR) for each environment. The MiniWoB tasks are designed so that rewards in the interval $(0, 1]$ indicate partial credit towards the task, while negative rewards indicate a failure. Given a list of rewards R , we thus compute the success rate as $\sum 1[R > 0] / \sum 1[R \neq 0]$. We can immediately evaluate two methods on all environments: 1) the random baseline, and 2) humans (refer to Figure 7).

Supervised Learning. We obtain a behavior cloning policy by training on the demonstrations using Adam (Kingma & Ba, 2014) with a learning rate of 10^{-3} and batch size of 32. We achieved better results by weighing click and keyboard event losses (which are rare compared to move events) 10 times higher in the objective. We then run the fixed policy on each environment for 100,000 steps (about 2 hours at 12FPS) and evaluate the success rate (see Figure 7, yellow bars).

Reinforcement Learning. We run 12 environments in parallel at 12 FPS for up to 1 million steps and perform an update every 200 time steps (i.e. training batches have size $12 \times 200 = 2400$ steps) with Adam and a learning rate of 10^{-4} . To mitigate the effects of our asynchronous setting, we train 3 times and use the best one. The results are shown in Figure 7 (green bars).

Interpretation. We summarize the quantitative results across all environments in Table 1. We refer to an environment as ‘‘Solved’’ if its success rate is at least half (50%) that of a human. From these numbers, it is evident that supervised learning slightly improves the policy (20.8% to 24.8%), but a much larger improvement can be obtained by fine-tuning the policy with reinforcement learning (24.8% to 34.8%). We also see that most of our performance comes

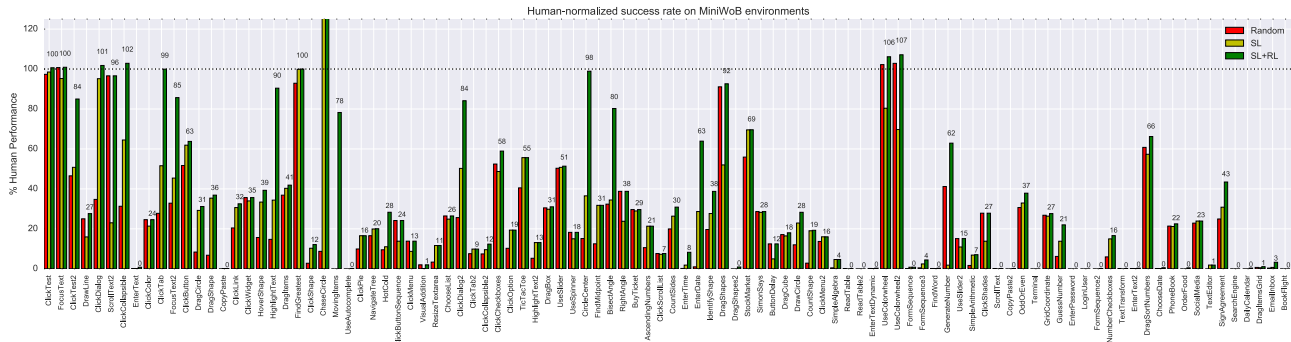


Figure 7. The success rate on all MiniWoB environments, normalized by human performance. (The performance on the cropped Chase-Circle is 250%). The tasks are arranged in an approximate/subjective level of difficulty from easy (left) to hard (right).

	Random	SL	SL+RL	#envs
Success Rate (SR)	20.8	24.8	34.8	100
% Solved	12	17	26	100
SR: Click	28.1	35.2	48.5	57
SR: Drag	21.5	24.7	34.6	15
SR: Mouse+Keyboard	6.4	3.5	7.8	21
SR: Compound	3.3	3.8	4.3	7

Table 1. Summary of MiniWoB human-normalized success rate across all environments, and across 4 environment categories based on the required interactions. #envs denotes the number of environments in each category. SL is supervised learning and RL is reinforcement learning.

from environments that require mouse interaction (Click / Drag). We also see a sharp drop in tasks that require keyboard input (7.8% SR). Finally, “Compound” environments are our most difficult environments (e.g. a synthetic email, flight booking, search engine, calendar, text editor, etc.); They combine multiple interactions over longer sequences (e.g. search for an email and reply with some text), and clearly pose a significant challenge (4.3% SR). Note that Random policy can do well in some environments because the action frequency is high (12 FPS), and our rewards for correct actions are scaled linearly based on time.

4.2. Results on Live Web Tasks (FormWoB)

Environment setup. Next, we evaluate our model on the four FormWoB tasks. The resolution of these environments is $375 \times 667 \times 3$. The FormWoB dataset contains four flight booking website: United (united.com), Alaska (alaskaair.com), JetBlue (jetblue.com) and American (aa.com). We run the environments at 1 frame per second to accommodate the load time of webpages.

Demonstration Data. For each website, we collected 100 (query, demonstration) pairs using AMT. Unlike MiniWoB, most of the interactions here involve clicking and typing. After preprocessing, each episode consists of approxi-

mately 30–50 keyboard or mouse events. Similar to MiniWoB, we divide the screen into 20×20 grid points, and use the key encoding scheme introduced in Section 3.1.

Model. Our model is the same 6-layer architecture in MiniWoB, except we remove the dragging actions. We also evaluate the LocalCNN model that directly outputs a $20 \times 20 \times 32$ dense feature map, which is used to drive attention and mouse clicks. We use a simple heuristic to combine the DOM together with the query to compute a query-specific *feature map*, which indicates salient locations in the input. In particular, we intersect the words in the query and the DOM using a similarity score based on edit distance, and “put” that score into the middle of the bounding box that contains that DOM element. For instance, if a query contains the word “From”, then any element in the webpage that contains the word “From” would have higher activation in the feature map.

We found that treating the keyboard simply as another categorical distribution was very challenging because the model would have to learn to type out entire phrases such as “San Francisco” one character at a time. Therefore, we augment the state with a pointer into each slot in the query and define actions for typing the next character of some slot. As an example, consider the following query with four slots:

k =	Departure City	S a n	F r a n c i s c o
	Destination City	N e w	Y o r k
	Departure Month	3	
	Departure Day	15	

In this example, we would have a multinomial distribution over the 4 slots. If the agent outputs the action sequence K1 K1 K1 K2 K2, it will first type ‘S’, ‘a’, ‘n’ (the prefix of “San Francisco”), reset the pointer for the first slot, and then type ‘N’, ‘e’.

Supervised Learning. We use similar supervised learning

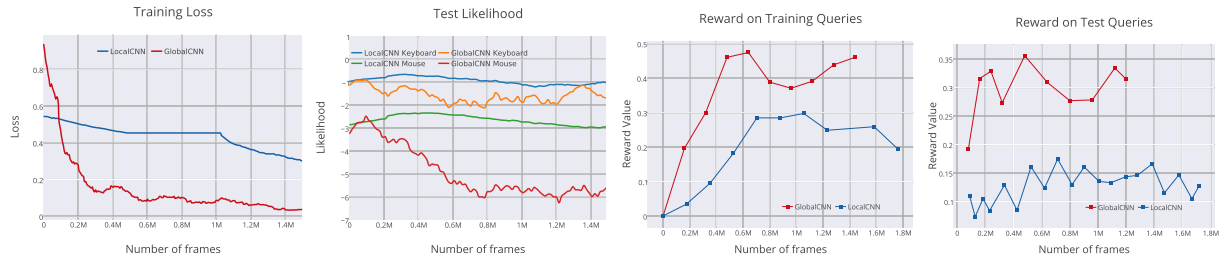


Figure 8. Learning curves and rewards on FormWoB task *Book a flight from (origin) to (destination) leaving (departure_date) and returning (return_date) on united.com*. GlobalCNN outperforms LocalCNN across the board.

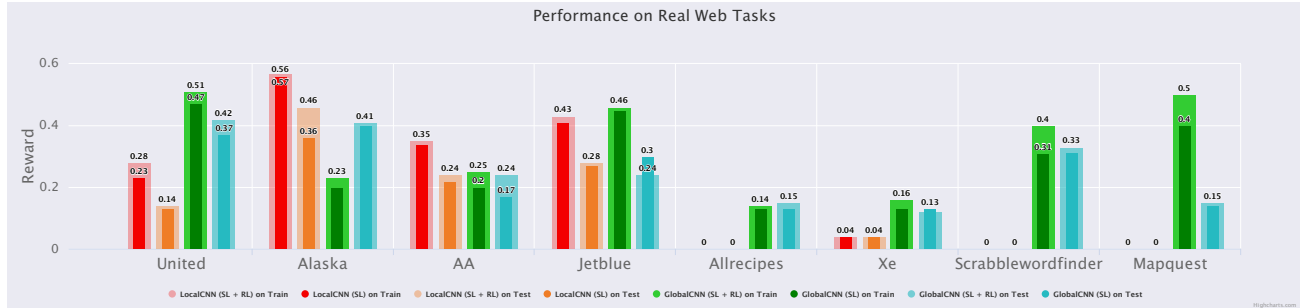


Figure 9. Rewards for the 4 FormWoB tasks and 4 QAWoB tasks. Random achieves 0, human is 1.

setting as in MiniWoB, except the learning rate is 10^{-4} and the keyboard event losses are weighted 20 times higher.

Reinforcement Learning. We fine-tune the models using RL on each of the environments separately. For every episode, we sample randomly from the set of queries and run the model at 8 FPS.

Evaluation. We are interested in measuring the model’s *generalization* ability across queries. We split the tasks on each website into 80% for training, and 20% for testing. First, we report the test likelihood as a metric to show how well the agent models human trajectories. We then evaluate the rewards the agent is able to achieve on both training and test sets. We report the average rewards over the final three checkpoints.

Results on FormWoB. Figure 8 shows the learning curves on the United website. The performance of random agents is identically zero on these tasks. Our model shows some learning and generalization. In particular, for flight booking, the model achieves 20%–30% of human level performance on training queries, and 16% on test queries. Figure 9 summarizes the model’s performance on 8 web tasks in our experiment.

We visualize the model’s attention output at some key frames in Figure 10. As we can see, the model generalizes by correctly selecting the city in dropdown and picking the correct date, aided by text matching. The CNN identifies

the “Submit” button even after some random scrolling has occurred. The most common failure mode is if the agent falls off the demonstrators’ state distributions (e.g. triggering an error message), it is difficult to take actions to recover.

4.3. Results on Crowdsourced Web Tasks (QAWoB)

Using same setup as in FormWoB, we perform experiments on the following websites from the QAWoB dataset: Xe (xe.com), Allrecipes (allrecipes.com), Scrabblewordfinder (scrabblewordfinder.org), and Mapquest (mapquest.org). The results of SL and SL+RL of both LocalCNN and GlobalCNN models on QAWoB are reported in Figure 9.

We find the performance of LocalCNN to be inadequate on these web tasks, while GlobalCNN performs much better. This is consistent with GlobalCNN achieving a lower training loss (~ 0.08) compared to LocalCNN (~ 0.2). It is likely that the inductive bias introduced in LocalCNN makes it incapable of fitting noisy human demonstrations. Figure 11(c) shows some example failure cases.

5. Related Work

Reinforcement learning environments. Our work enjoys the company of many recent projects that aim to provide challenging environments for reinforcement learning agents, including the ATARI Learning Environment (Belle-

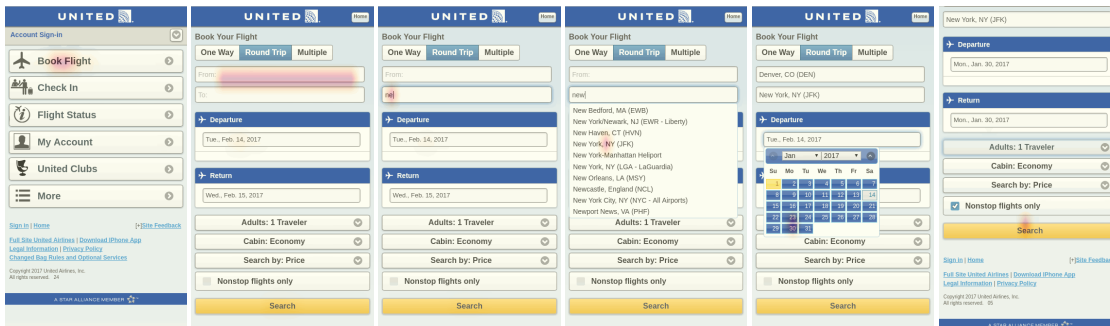


Figure 10. Visualization of agent’s distribution over mouse locations on a test task from FormWoB (united.com)

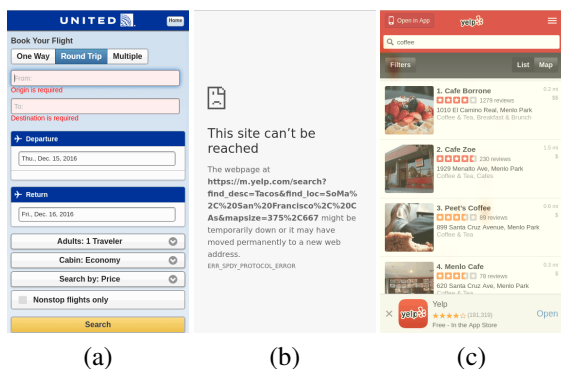


Figure 11. Common error cases: (a) Deviating from the human demonstration state distribution. (b) HTTP request not in our cache. (c) Picking the wrong answer to the query.

mare et al., 2013), MuJoCo (Todorov et al., 2012), ComAI (Baroni et al., 2017), Project Malmö (Johansson et al., 2016), SNES (Bhonker et al., 2016), TorchCraft (Synnaeve et al., 2016), DeepMind Lab (Beattie et al., 2016) and ViZ-Doom (Kempka et al., 2016). World of Bits differs primarily by its focus on the open-domain realism of the web.

Performing tasks on the web. The web is a rich environment with a long tail of different phenomena and the emergence of high-level semantics. The information retrieval and natural language processing communities have long used the web as a source of textual data (Hearst, 1992; Brill et al., 2002; Etzioni et al., 2005). Nogueira & Cho (2016) introduced WebNav, a software tool that transforms a website into a synthetic goal-driven web navigation task. Some work has also focused on mapping natural language queries to programs that operate on the DOM structure of web pages (Pasupat & Liang, 2014). These previous works focus on higher-level actions that abstract away the visual layout and the keyboard and mouse movements, which limits their scope, especially given the increasing prevalence of highly interactive websites. To our knowledge, our work is

the first to tackle the problem of interacting with websites using both vision and raw mouse and keyboard actions on open-domain tasks at scale.

Natural language to actions. There is a large body of work on connecting language to actions. Closely related to our work is Branavan et al. (2009), who used reinforcement learning to map instructions (e.g. a Windows troubleshooting article) to actions over a user interface in a virtual machine; however, they used preprocessed actions. Other work operates in the context of navigation (Vogel & Jurafsky, 2010; Tellex et al., 2011; Artzi & Zettlemoyer, 2013), and building tasks (Long et al., 2016; Wang et al., 2016). The focus of these efforts is on modeling natural language semantics. Our work provides a bridge between this semantic-oriented work and the more control-oriented tasks found in most reinforcement learning environments.

6. Conclusion

In this paper, we introduced World of Bits (WoB), a platform that allows agents to complete web tasks with keyboard and mouse actions. Unlike most existing reinforcement learning platforms, WoB offers the opportunity to tackle realistic tasks at scale. We described a progression of three techniques to create web tasks suitable for reinforcement learning: 1) Minimalistic tasks such as Mini-WoB (hand-crafted tasks), 2) Proxy environments such as FormWoB (live websites, hand-crafted tasks), and 3) Crowdsourced environments such as QAWoB (live websites, crowdsourced tasks). Finally, we showed that while standard supervised and reinforcement learning techniques can be applied to achieve adequate results across these environments, the gap between agents and humans remains large, and welcomes additional modeling advances.

Acknowledgements

This work was done in collaboration between OpenAI and Stanford. Tim Shi is partly funded by Tencent. We would like to thank John Schulman for insightful discussions.

References

- Artzi, Y. and Zettlemoyer, L. Weakly supervised learning of semantic parsers for mapping instructions to actions. *Transactions of the Association for Computational Linguistics (TACL)*, 1:49–62, 2013.
- Bahdanau, D., Cho, K., and Bengio, Y. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.
- Baroni, Marco, Joulin, Armand, Jabri, Allan, Kruszewski, Germàn, Lazaridou, Angeliki, Simonic, Klemen, and Mikelov, Tomas. Commai: Evaluating the first steps towards a useful general ai. *arXiv preprint arXiv:1701.08954*, 2017.
- Beattie, Charles, Leibo, Joel Z, Teplyashin, Denis, Ward, Tom, Wainwright, Marcus, Küttler, Heinrich, Lefrancq, Andrew, Green, Simon, Valdés, Víctor, Sadik, Amir, et al. Deepmind lab. *arXiv preprint arXiv:1612.03801*, 2016.
- Bellemare, Marc G, Naddaf, Yavar, Veness, Joel, and Bowling, Michael. The arcade learning environment: An evaluation platform for general agents. *J. Artif. Intell. Res.(JAIR)*, 47:253–279, 2013.
- Bhonker, Nadav, Rozenberg, Shai, and Hubara, Itay. Playing snes in the retro learning environment. *arXiv preprint arXiv:1611.02205*, 2016.
- Branavan, Satchuthanathavale RK, Chen, Harr, Zettlemoyer, Luke S, and Barzilay, Regina. Reinforcement learning for mapping instructions to actions. In *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP: Volume 1-Volume 1*, pp. 82–90. Association for Computational Linguistics, 2009.
- Brill, E., Dumais, S., and Banko, M. An analysis of the AskMSR question-answering system. In *Association for Computational Linguistics (ACL)*, pp. 257–264, 2002.
- Brockman, Greg, Cheung, Vicki, Pettersson, Ludwig, Schneider, Jonas, Schulman, John, Tang, Jie, and Zaremba, Wojciech. Openai gym. *arXiv preprint arXiv:1606.01540*, 2016.
- Deng, J., Dong, W., Socher, R., Li, L., Li, K., and Fei-Fei, L. ImageNet: A large-scale hierarchical image database. In *Computer Vision and Pattern Recognition (CVPR)*, pp. 248–255, 2009.
- Etzioni, O., Cafarella, M., Downey, D., Popescu, A., Shaked, T., Soderland, S., Weld, D. S., and Yates, A. Unsupervised named-entity extraction from the web: An experimental study. *Artificial Intelligence*, 165(1):91–134, 2005.
- Hearst, M. A. Automatic acquisition of hyponyms from large text corpora. In *International Conference on Computational Linguistics*, pp. 539–545, 1992.
- Johansson, F., Shalit, U., and Sontag, D. Learning representations for counterfactual inference. In *International Conference on Machine Learning (ICML)*, 2016.
- Kempka, Michał, Wydmuch, Marek, Runc, Grzegorz, Toczek, Jakub, and Jaśkowski, Wojciech. Vizdoom: A doom-based ai research platform for visual reinforcement learning. *arXiv preprint arXiv:1605.02097*, 2016.
- Kingma, Diederik and Ba, Jimmy. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.
- Levine, Sergey, Pastor, Peter, Krizhevsky, Alex, and Quillen, Deirdre. Learning hand-eye coordination for robotic grasping with deep learning and large-scale data collection. *arXiv preprint arXiv:1603.02199*, 2016.
- Long, R., Pasupat, P., and Liang, P. Simpler context-dependent logical forms via model projections. In *Association for Computational Linguistics (ACL)*, 2016.
- Mikelov, Tomas, Joulin, Armand, and Baroni, Marco. A roadmap towards machine intelligence. *arXiv preprint arXiv:1511.08130*, 2015.
- Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., et al. Human-level control through deep reinforcement learning. *Nature*, 518 (7540):529–533, 2015.
- Mnih, Volodymyr, Badia, Adria Puigdomenech, Mirza, Mehdi, Graves, Alex, Lillicrap, Timothy P, Harley, Tim, Silver, David, and Kavukcuoglu, Koray. Asynchronous methods for deep reinforcement learning. In *International Conference on Machine Learning*, 2016.
- Nogueira, Rodrigo and Cho, Kyunghyun. End-to-end goal-driven web navigation. In *Advances in Neural Information Processing Systems*, pp. 1903–1911, 2016.
- Pasupat, P. and Liang, P. Zero-shot entity extraction from web pages. In *Association for Computational Linguistics (ACL)*, 2014.
- Pinto, Lerrel and Gupta, Abhinav. Supersizing self-supervision: Learning to grasp from 50k tries and 700 robot hours. In *Robotics and Automation (ICRA), 2016 IEEE International Conference on*, pp. 3406–3413. IEEE, 2016.

- Pomerleau, Dean A. Alvin, an autonomous land vehicle in a neural network. Technical report, Carnegie Mellon University, Computer Science Department, 1989.
- Rajpurkar, P., Zhang, J., Lopyrev, K., and Liang, P. Squad: 100,000+ questions for machine comprehension of text. In *Empirical Methods in Natural Language Processing (EMNLP)*, 2016.
- Schulman, John, Levine, Sergey, Abbeel, Pieter, Jordan, Michael I, and Moritz, Philipp. Trust region policy optimization. In *ICML*, pp. 1889–1897, 2015a.
- Schulman, John, Moritz, Philipp, Levine, Sergey, Jordan, Michael, and Abbeel, Pieter. High-dimensional continuous control using generalized advantage estimation. *arXiv preprint arXiv:1506.02438*, 2015b.
- Silver, David, Huang, Aja, Maddison, Chris J, Guez, Arthur, Sifre, Laurent, Van Den Driessche, George, Schrittwieser, Julian, Antonoglou, Ioannis, Panneershelvam, Veda, Lanctot, Marc, et al. Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016.
- Sutton, R., McAllester, D., Singh, S., and Mansour, Y. Policy gradient methods for reinforcement learning with function approximation. In *Advances in Neural Information Processing Systems (NIPS)*, 1999.
- Synnaeve, Gabriel, Nardelli, Nantas, Auvolat, Alex, Chintala, Soumith, Lacroix, Timothée, Lin, Zeming, Richoux, Florian, and Usunier, Nicolas. Torchcraft: a library for machine learning research on real-time strategy games. *arXiv preprint arXiv:1611.00625*, 2016.
- Tellex, S., Kollar, T., Dickerson, S., Walter, M. R., Banerjee, A. G., Teller, S. J., and Roy, N. Understanding natural language commands for robotic navigation and mobile manipulation. In *Association for the Advancement of Artificial Intelligence (AAAI)*, 2011.
- Todorov, Emanuel, Erez, Tom, and Tassa, Yuval. Mujoco: A physics engine for model-based control. In *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*, pp. 5026–5033. IEEE, 2012.
- Vogel, A. and Jurafsky, D. Learning to follow navigational directions. In *Association for Computational Linguistics (ACL)*, pp. 806–814, 2010.
- Wang, S. I., Liang, P., and Manning, C. Learning language games through interaction. In *Association for Computational Linguistics (ACL)*, 2016.