Autonomous Web Navigation with RL

Chinmoy Samant, cs59688
Sagnik Majumder, sm72878

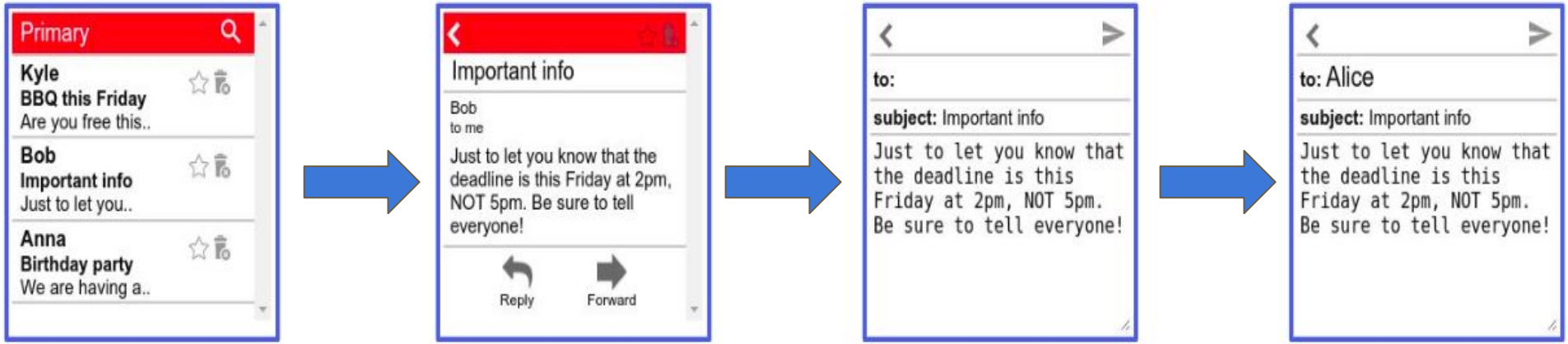
Problem Description

- In the Autonomous Web Navigation task, an agent is trained to navigate an interactive web environment to complete a certain query-based web task.
- The query can be a structured query or a natural language query.
- Examples of web tasks include:
  - Forwarding an email from inbox
  - Entering data (like name, date, time, place, etc.) on a webpage

Examples of web tasks

- Forwarding an email from inbox

Example - *forward* an email from *Bob* to *Alice* => goal - {**task:** forward, **from:** Bob, **to:** Alice}

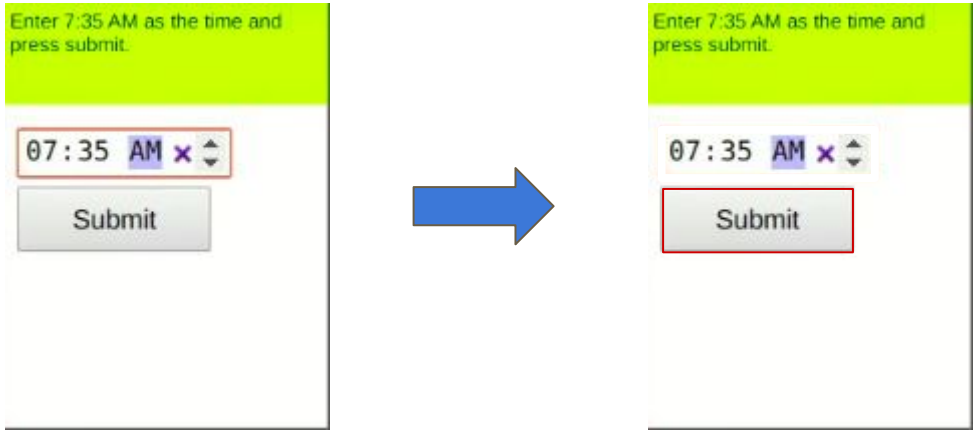


Liu et al. (2018), Shi et al. (2017)

Examples of web tasks continued ...

- Entering time on a webpage

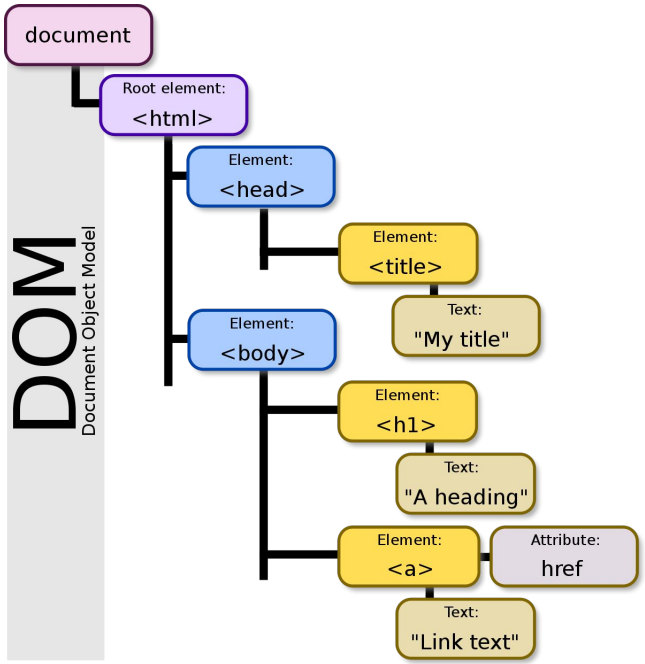
Example - Enter *7:35 am* and *submit* => goal - {**task:** enter-time, **value:** 7:35 am}



Liu et al. (2018), Shi et al. (2017)

Common characteristics of web tasks

- **High-dimensional environment:** numerous constituent elements in DOM-tree representation of webpage, makes the environment very high dimensional



Wikipedia

Common characteristics continued ...

- **Sparse rewards:** a web-navigation agent gets a reward (success) only if it completes the task perfectly within an allotted time, due to complex environment, navigation trajectories can be long which aggravate sparsity

- **Variety in webpage type and content:** variety in webpages makes it difficult to hand-engineer navigation agents for each different page existing on the internet, necessitates the design of a model that can learn a task on its own

Challenges in naively using RL

- **Sparse rewards**: standard RL algorithms like Q-learning, SARSA etc. fail when rewards are so sparse

- **High-dimensional state-action space**: impossible to supply enough trajectories that have broad-enough coverage, so direct behavioral cloning (BC) overfits and fails

- **Multi-modal data**: web data is a mix of structured(e.g. HTML) and unstructured inputs (e.g. natural language and images) unlike other common RL domains like game-playing, robotics etc.

Liu et al. (‘18), Shi et al. (‘17)

Solution - Workflow Guided Exploration (WGE)

- Train an additional policy that constrains exploration around demos through use of workflows, restrict possible actions given a state

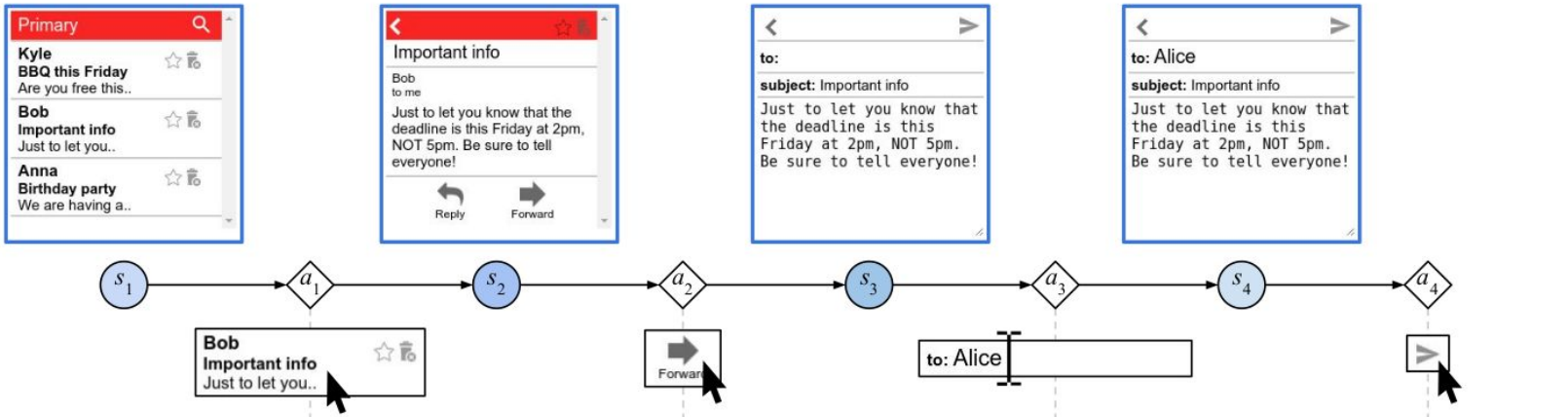
- Workflows provide high-level actions that are conditional on the demonstration but independent of the state (e.g., in *email-forward*, type into the ‘to’ field instead of what to type exactly)

- Leverage meta-knowledge of the task to draw a rough layout of actions and follow it blindly (environment-blindedness!)

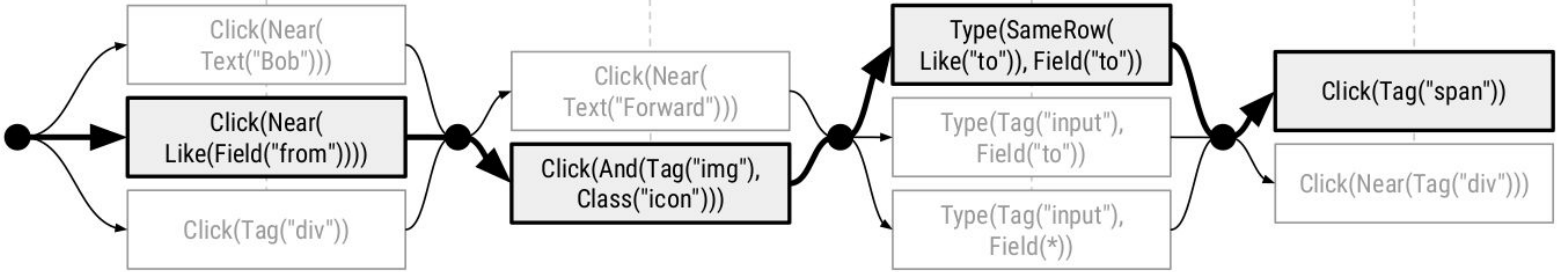
Liu et al. (2018), Deka et al. (2017)

Demonstration to workflow lattice

Demonstration: goal = {**task:** forward, **from:** Bob, **to:** Alice}

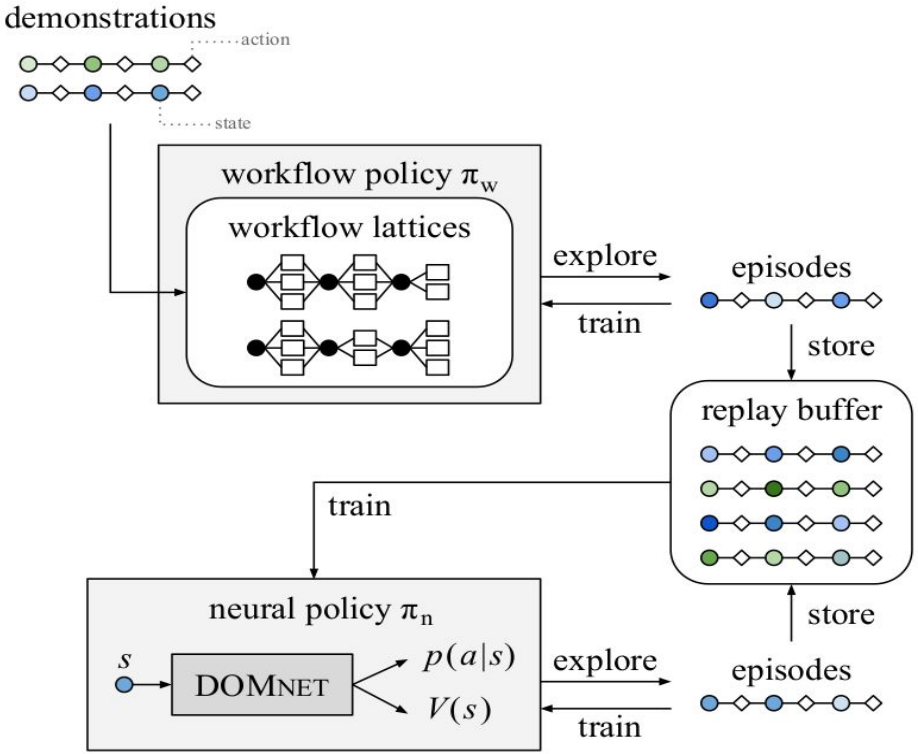


Workflow lattice:



Liu et al. (2018)

Training steps and pseudocode



Preprocessing:

for all demonstrations d **do**
  Induce workflow lattice from d

Every iteration:

Observe an initial environment state
π_w samples a workflow from a lattice
Roll out an episode e from the workflow
Use e to update π_w
if e gets reward $+1$ **then**
  Add e to replay buffer

Periodically:

if replay buffer size > threshold **then**
  Sample episodes from replay buffer
  Update π_n with sampled episodes
Observe an initial environment state
π_n rolls out episode e
Update π_n and critic V with e
if e gets reward $+1$ **then**
  Add e to replay buffer

Liu et al. (2018)

Workflow policy

- A single-layered neural network with softmax outputs that decides the weights for every workflow step (e.g., weights for Click(Near(Text('Forward'))) and Click(And(Tag('img'), Class('icon'))), they sum to 1)

$$z_t \sim \pi_w(z|d,t) \propto exp(\psi_{z,t,d})$$

- Given a workflow step, the workflow policy samples an action from a uniform distribution with a probability equal to the inverse of the number of possible actions given that workflow step

$$a_t \sim p(a|z_t, s_t) = \frac{1}{|z_t(s_t)|}$$

- Workflow policy is trained with REINFORCE with a constant baseline of 0.1

Liu et al. (2018), Williams (1992), Sutton (1998)

Neural policy

- DOMNet architecture to leverage both spatial and hierarchical information in a DOM-tree; embedder module for embedding DOM-tree and query/instruction and attention module

- Outputs policy and an approximated value function

- Trained using synchronous advantage actor-critic (A2C)

Liu et al. (2018), Mnih et al. (2016)

Reward shaping - extension to baseline

- Replace hard reward (reward only at the end of an episode) with a time-dependent soft reward (reward augmentation) => reduction in reward sparsity, might make learning easier and more efficient

- Define a state-dependent potential function that counts the fraction of goal achieved depending on the number of steps in goal that are completed, and take the weighted difference of potential between two consecutive states to compute the soft reward

$$R_{potential} = \gamma(Potential(s_{t+1}, g) - Potential(s_t, g))$$

Ng et al. (1999), Gur et al. (2018)

Dataset, tasks and experiments

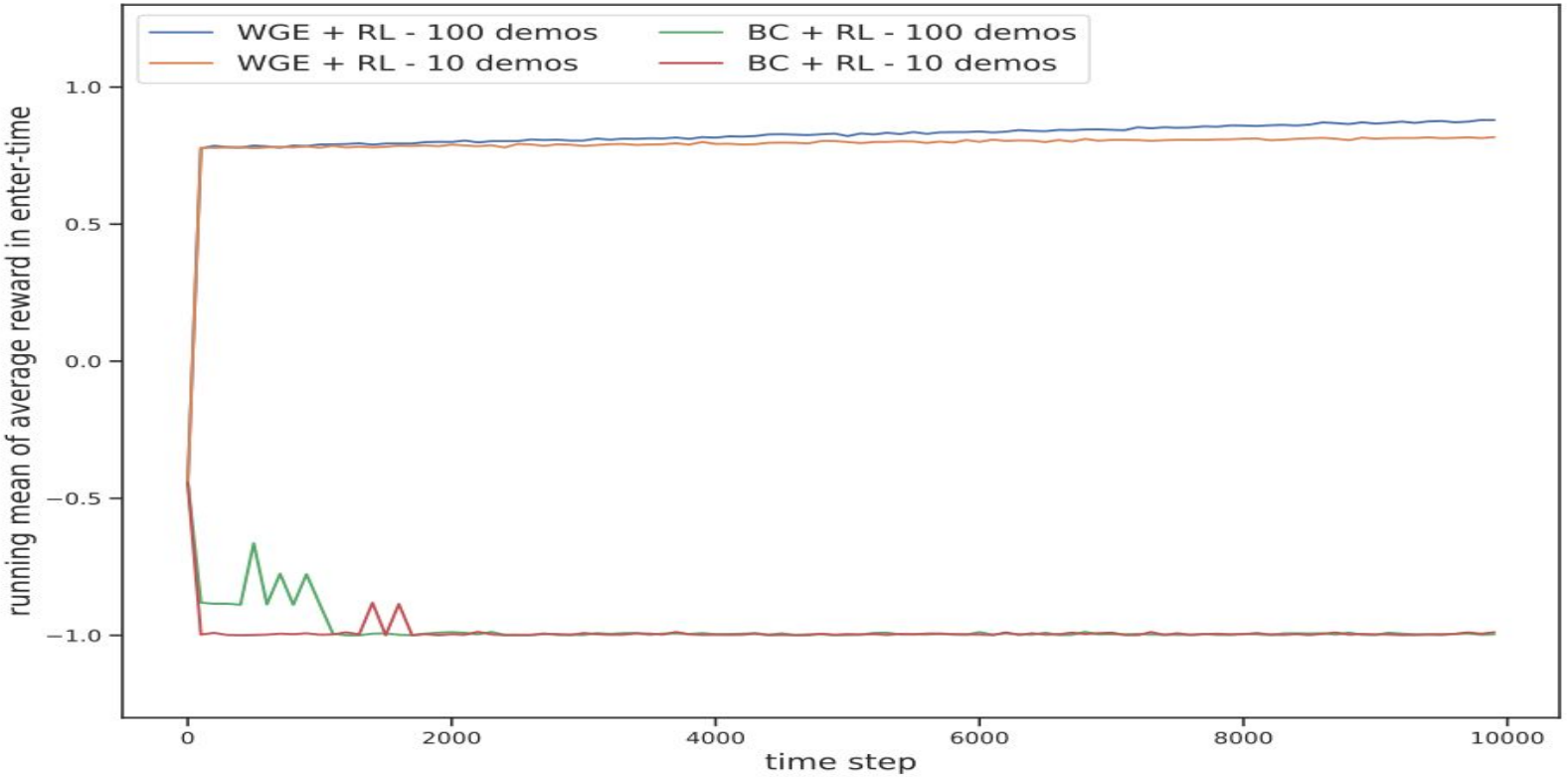
- MiniWoB++ dataset (extension of MiniWoB) with over 105 interactive web tasks

- Experiment with two prototypical tasks from MiniWoB++
  - **Email-inbox:** basic tasks like email forwarding, starring or deleting email etc. with both structured and natural language queries
  - **Enter-time:** enter a given time on a webpage

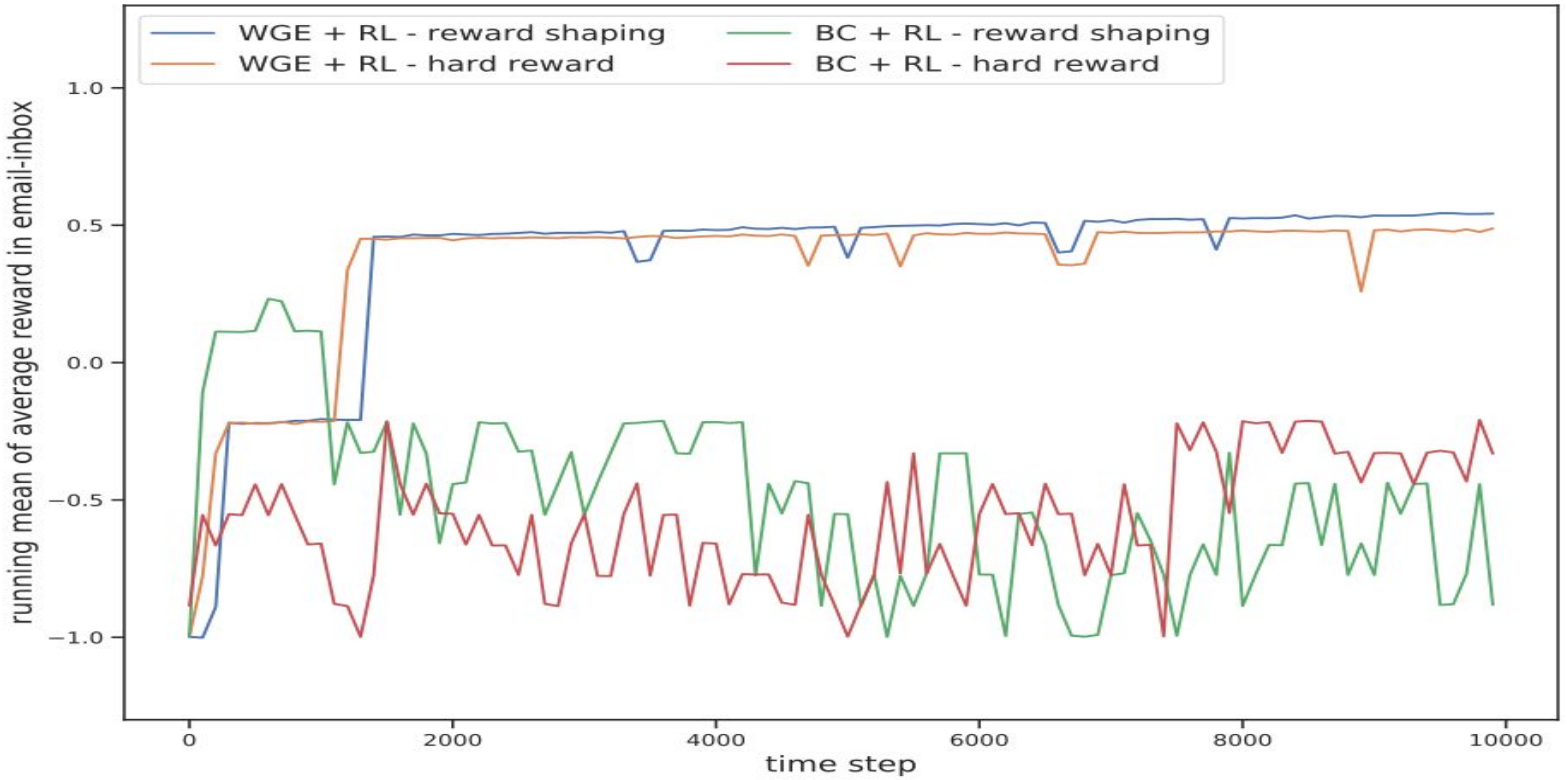
- Experiments with different number of demonstrations and reward shaping, and comparison of WGE with BC

Liu et al. (‘18), Shi et al. (‘17)

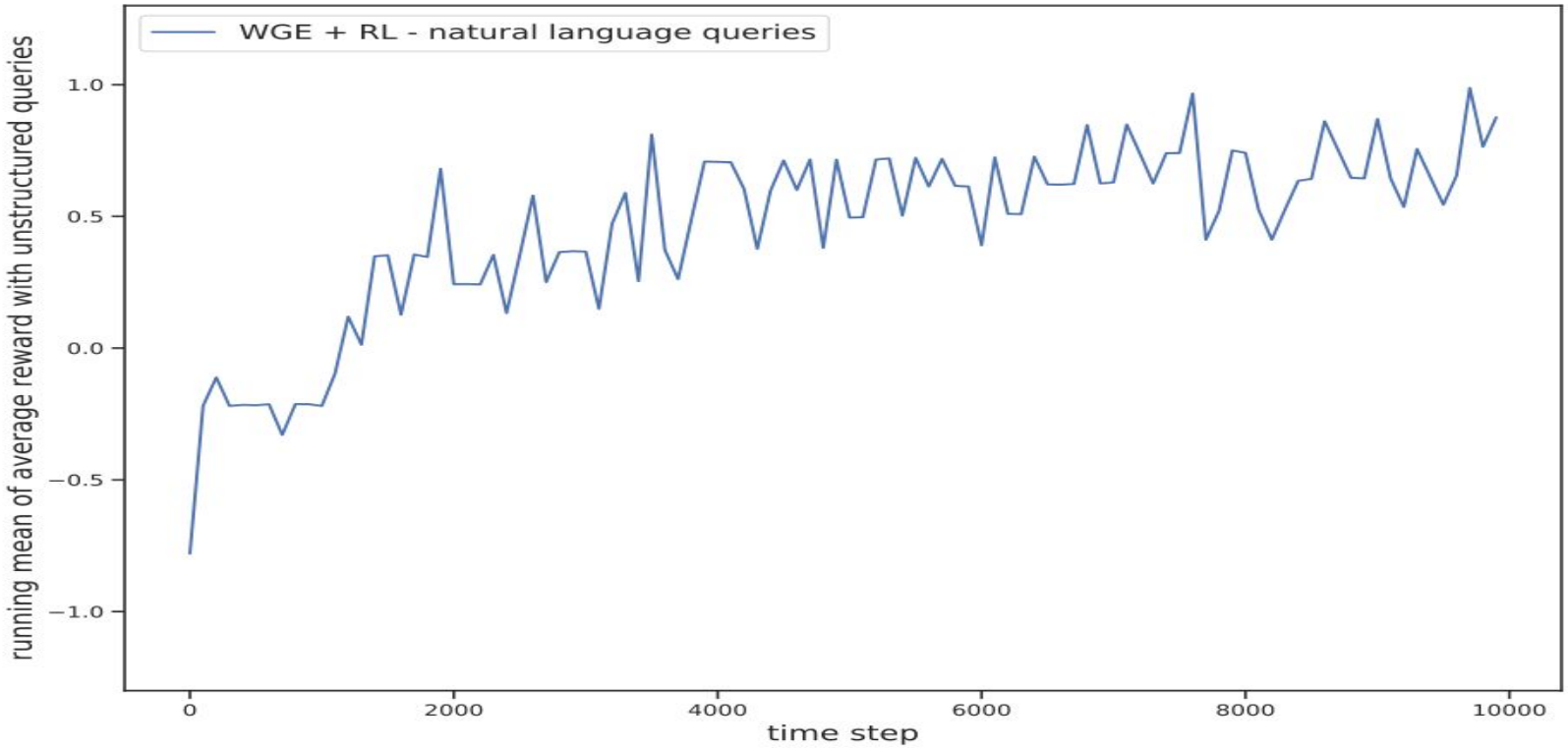
WGE vs BC; effect of no. of demos on enter-time



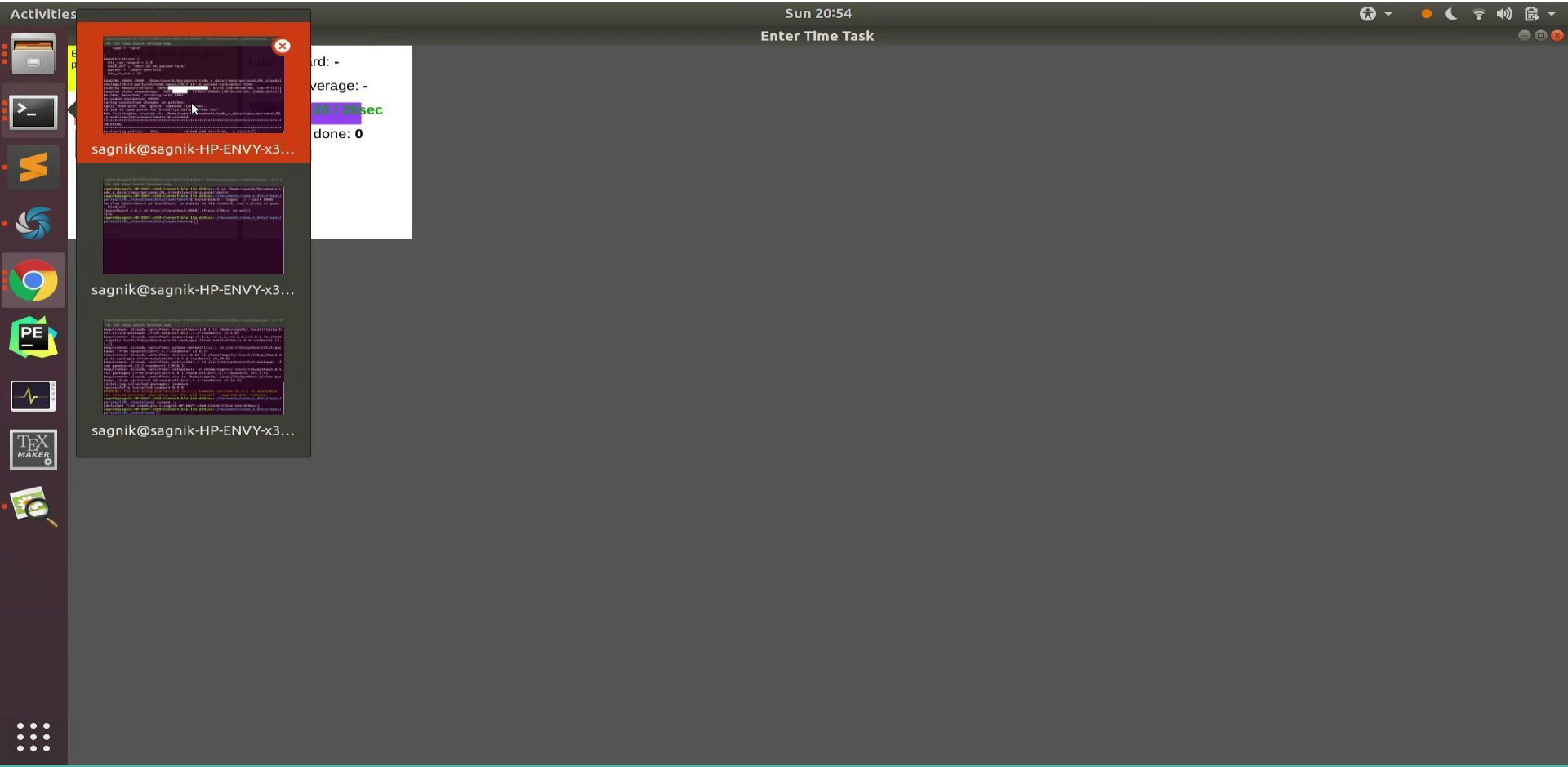
Effect of no. of reward shaping on email-inbox



Email-inbox with natural language query



Short demo of WGE on enter-time



Short demo of WGE on email-inbox

Conclusion

- WGE far outperforms BC for automated web navigation task irrespective of the number of demos provided

- Reward shaping asymptotically shows performance improvement over vanilla WGE algorithm

- It is possible for an agent to learn web tasks even from natural language queries if it’s equipped with a powerful-enough natural language parser

Future Work

- Study the effect task-specific workflow lattices instead of demonstration-specific lattices on the performance of WGE, should allow model to learn richer information about the task and generalize better

- Try curriculum learning for ‘warm starting’ the agent. Curriculum learning can initially place the agent very close to the goal and lead to efficient learning

- Benchmark WGE and reward shaping algorithms on other web-based tasks from MiniWoB++

Thank you!

For any questions/suggestions, please send an email to:

Chinmoy Samant, cs59688
chinmoy@cs.utexas.edu

Sagnik Majumder, sm72878
sagnik@cs.utexas.edu