OSWorld Setup Tutorial

1. VM Environment Setup

Please be careful about whether the following commands/steps are conducted on the VM running instance or your host machine.

Prerequisites

1. Download and install the latest version of VMWare on the host machine
   - For Windows/Linux, install VMWare Workstation Pro
   - For MacOS, install VMWare Fusion
   - Serial numbers can be found online
2. Download and install Ubuntu 22.04 LTS image on VMWare
   - Download link: https://cdimage.ubuntu.com/jammy/daily-live/current/
   - Pay attention to the machine architecture of the host (amd64 or arm64). For **arm64** architecture (such as MacOS), please install the **aarch64** version
   - During installation, allocate at least **50G** disk space, **create an account with the name *user*** (the home directory should be */home/user/*) **and password *'password'* (' not included)** and choose English as the default language
   - About GUI display, please ensure **Xorg (X11)** is chosen instead of wayland
     - **This step is significant to UI operations such as click, press and scroll!**
     - Tutorial on how to check and change the GUI display

Install Dependencies on VM

1. Firstly, **in the VM running instance**, open the terminal and install VM tools:
   - This allows you to manipulate VM instances through shell commands in the host

```
sudo apt-get install open-vm-tools
```

2. Then, **in the host machine**, clone the OSWorld git repository and copy the entire *desktop_env/server* directory to the running VM. For example, we can use the *vmrun* CopyFile command:
   - [abspath_to_vmx_file] is the absolute path to the *.vmx* file, e.g.,
     - on Windows, it should be like: *C:\Users\tianbaox\Documents\Virtual_Machines\Ubuntu\Ubuntu.vmx*
     - on MacOS, it should be like: */Users/rhythmcao/Virtual_Machines.localized/ubuntu.vmwarevm/ubuntu.vmx*
   - for more vmrun commands, see VMWare official doc for reference

```
vmrun -T ws -gu user -gp password CopyFileFromHostToGuest
[abspath_to_vmx_file] desktop_env/server /home/user/server
```

2. Next, **in the VM running instance**, install the following dependencies:
   - apt-get install:
     - python3-pip, python3-tk, python3-dev
     - git, vim, jq, curl, wmctrl, ffmpeg, socat, net-tools

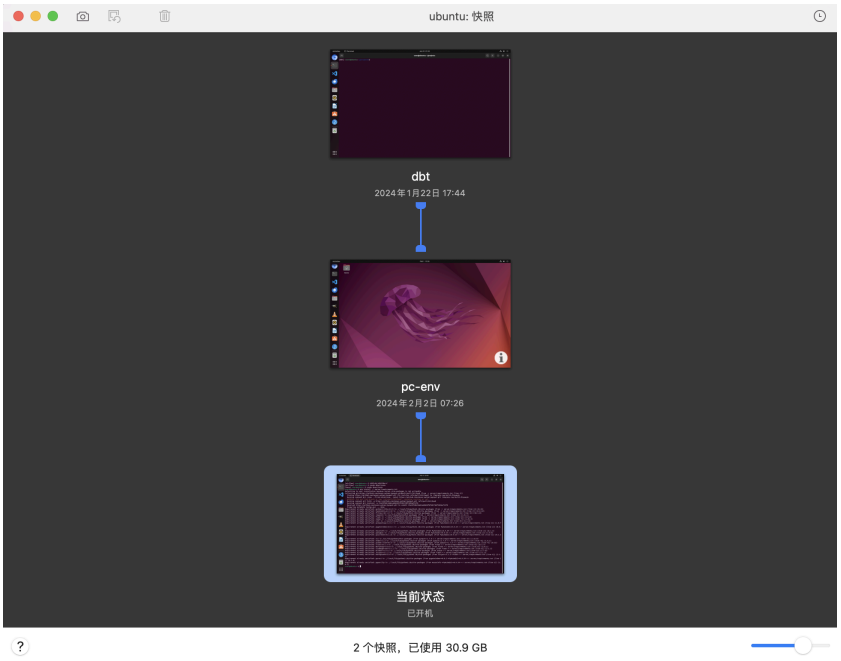
    - gnome-screenshot, accerciser
- pip3 install -r ~/server/requirements.txt

3. Start the Flask service in the backend of the **running VM instance**:
    - **Please ensure this command is executed before saving VM snapshots (next Section)**

```
cd /home/user/server/
# you can close the terminal window after running the following command
nohup python3 -u main.py > out.log 2>&1 &
```

VM Snapshots

- What is a VM snapshot?
    a. VMWare uses snapshot trees to manage different states. We can save and revert to the complete state of one VM running instance. (The checklist of all snapshots can be found in the *.vmsd* file under the same directory as *.vmx*)



- When you work on Spider2.0 GUI, please follow steps below:
    a. Install the following tools in the VM instance:
        - VSCode (take care of platform infos, e.g., amd64 or arm64)
        - Google Chrome or Chromium (Google Chrome is not available on Mac-OS hosted Ubuntu, you can only install Chromium instead)
            - ensure that the browser can be launched via terminal command `*google-chrome*` or `*chromium-browser*`
        - anaconda3, the installed path is `/home/user/anaconda3`
        - docker, ensure that the current user is added into docker group to avoid sudo (you need to restart the VM machine)
        - TO BE CONTINUED: other tools
    b. To avoid being interrupted by warning or pop-up alerts during debugging or testing, disable the following functions in VM:
        - Login in and auto-translation reminder for Chromium/Chrome

        - Disable screen lock/protection for Ubuntu system
        - Auto-update reminder of Ubuntu software mangement
        - Disable the “unlock login keyring” prompt: change password to empty
        - TO BE CONTINUED: other operations
    c. Take a snapshot and name it. This name will be used in the “*snapshot*” field in one data example to revert the VM instance to a prepared state

Running Test

- Take one snapshot and name it as you like, e.g., “*chrome*”.
- Run the following script in the root directory of DesktopEnv on your host machine:
    - replace [xxx] with concrete values
    - if error “python is not found” is raised, create a symbolic link “python” for “python3” in VM and save the snapshot (note that adding an alias python=python3 in *~/.bashrc* is useless)

```
conda create -n [env_name] python=3.11 -y && conda activate [env_name]
pip install -r requirements.txt
playwright install chromium
python main.py -p [abspath_to_vmx_file] -s [snapshot_name]
```

2. Format of Data Fields

A typical annotated data example *.json* dict: (partly simplified)

```
{
 "id": "6a33f9b9-0a56-4844-9c3f-96ec3ffb3ba2", # unique example id
 "snapshot": "libreoffice_writer", # name of VM snapshot to revert to
 "instruction": "Please help me remove all highlighted words.",
 "source": ["url_link_to_the_source_that_this_example_comes_from"],
 "counterpart": "xxxx-xxxx-xxxx-xxxx-xxxx", # the verbose example id of the current
abstract example, or vice versa
 "config": [
   {
     "type": "download",
     "parameters": {
       "files": [
         {
           "url": "google_drive_link_to_download_the_initial_file",
           "path": "/home/user/Desktop/sample-recruitment-phone-script.odt"
         }
       ]
     }
   }
 ],
 "related_apps": ["libreoffice_writer"],
```

```
 "tags": ["cli", "abstract", "account"],
 "evaluator": {
   "postconfig": [
     {
       "type": "activate_window",
       "parameters": {
         "window_name": "sample-recruitment-phone-script.odt - LibreOffice Writer",
         "strict": true
       }
     }
   ],
   "func": "check_highlighted_words",
   "expected": {
     "type": "cloud_file",
     "path": "google_drive_link_to_download_the_golden_file",
     "dest": "sample-recruitment-phone-script_Gold.odt"
   },
   "result": {
     "type": "vm_file",
     "path": "/home/user/Desktop/sample-recruitment-phone-script.odt",
     "dest": "sample-recruitment-phone-script.odt"
   },
   "options": {}
 }
}
```

Among all fields, the most significant is **instruction**, **config** and **evaluator**.

instruction field

This field is exactly the user input which tells the agent what to resolve. Notice that the instruction should be natural, realistic and clear. It comes from online realistic scenarios and reflects real-world user intention. For easy setup and precise evaluation, it can be slightly adapted. See Spider2.0 GUI: Write Task Instructions on how to write task instructions.

config field

This field defines how the environment is set up or prepared for each example. It is a list of dict. Each dict invokes one setup function to prepare the initial VM environment.

- Functions are invoked sequentially according to the list order.
- All functions are defined in "*desktop_env/controllers/setup.py*", e.g.,
    - in the example above, dict with type="*download*" will call the setup function below to download and upload initial files to the VM machine
- **UPDATE**: Now, we add one package "*desktop_env/configs*" for easier config extension. Remember to add the suffix "_setup" for your customized config function and import it in "*desktop_env/configs/__init__.py*".

```
def _download_setup(self, files: List[Dict[str, str]]):
```

```
    """
    Args
        files (List[Dict[str, str]]): files to download. list of dict like
            {
                "url": str, the url to download
                "path": str, the path on the VM to store the downloaded file
            }
    """
```

evaluator field

This complex field defines how the predicted/golden result is obtained and how the automatic evaluation is conducted. It can be further composed into 5 parts:

postconfig field

This field is similar to *config* setup. The only difference is that each operation in the list is applied after the agent/human has completed the task.

- For example, in the example above, the "*activate_window*" function makes the specified "*window_name*" the current activated/focused window in the desktop
- It reuses setup functions defined in "*desktop_env/controllers/setup.py*"

```
{
  "type": "activate_window",
  "parameters": {
    "window_name": "sample-recruitment-phone-script.odt - LibreOffice Writer",
    "strict": true
  }
}
```

func field

This field defines the metric name that will be invoked to obtain the result (usually 0 or 1). Different examples may use quite different functions. All available metrics are defined in "*desktop_env/evaluator/metrics/__init__.py*". In the example above, it refers to the function "*check_hightlighted_words*":

```
def check_highlighted_words(file_path1, file_path2):
    if not compare_docx_files(file_path1, file_path2):
        return 0

    doc = load(file_path1)
    highlighted = False

    for span in doc.getElementsByType(Span):
        style_name = span.getAttribute('stylename')
        if style_name:
            for automatic_style in doc.automaticstyles.childNodes:
                if automatic_style.getAttribute('name') == style_name:
```

```
                    for property in automatic_style.childNodes:
                        if property.getAttribute('backgroundcolor') == '#ffff00':
                            highlighted = True
                            break
            if highlighted:
                break

    return 0 if highlighted else 1
```

result field

This field is used to extract the expected/predicted “output”. The output can be a literal string, structured file or some ending states in the running VM. The “*type*” field of the dict gives the function name to retrieve the result. The checklist of all functions is provided in “*desktop_env/evaluators/getters/__init__.py*”. In the example above, it calls the “*get_vm_file*” function to copy the desired file from the VM to a local file path.

```
"result": {
  "type": "vm_file",
  "path": "/home/user/Desktop/sample-recruitment-phone-script.odt",
  "dest": "sample-recruitment-phone-script.odt"
}
```

expected field

This field is similar to the “*result*” field with the only difference that it is used to obtain the golden reference, e.g., an output file, expected state or literal answer. It also reuses the functions in “*desktop_env/evaluators/getters/__init__.py*”. In the example above, it calls the “*get_cloud_file*” function to download the gold file from the google drive link to a local path. Notice that, the fetched outputs of both “*result*” and “*expected*” fields serve as inputs of the metric function in the “*func*” field.

```
"expected": {
   "type": "cloud_file",
   "path": "google_drive_link_to_download_the_golden_file",
   "dest": "sample-recruitment-phone-script_Gold.odt"
}
```

options field

This field provides the keyword arguments for the metric function “*func*”. Key-values in this field do not depend on the predicted or golden output. It is optional and will be set to an empty dict if not provided.

```
metric_func(result_state, expected_state, **options)
```

Evaluation Extension

To support extra hard evaluation and encourage re-use of function modules (e.g., some examples may need to invoke multiple metric functions and only all results are correct, the task is successful), the original “*evaluator*” field is extended to support logical operators

“and”/“or”. That is, values of the following 4 fields (“*func*”, “*result*”, “*expected*” and “*options*”) are extended from a single string/dict to a list of the same length.

- “*conj*” is chosen from [“and”, “or”] and will be set to “and” if not provided

```
"evaluator": {
   "func": ["func1", "func2", "func3"],
   "conj": "and",
   "result": [{"type": ""}, {"type": ""}, {"type": ""}],
   "expected": [{"type": ""}, {"type": ""}, {"type": ""}],
   "options": [{}, {}, {}] # if not provided, will be set to a list containing only
empty dicts with the same length as other fields
}
```

To sum up:

- both “*config*” and “*evaluator->postconfig*” fields will search setup functions in “*desktop_env/controllers/setup.py*” to set up and post-process the environment
- evaluation metrics are defined in the field “*evaluator->func*”, which can be retrieved in “*desktop_env/evaluators/metrics/__init__.py*”
- “*evaluator->result*” and “*evaluator->expected*” fields will invoke functions in “*desktop_env/evaluators/getters/__init__.py*” to obtain the expected and golden outputs respectively

Remember: reuse defined functions whenever possible, but feel free and be creative to add necessary functions if unavoidable.

```
def check_highlighted_words(file_path1, file_path2):
   if not compare_docx_files(file_path1, file_path2):
       return 0


   doc = load(file_path1)
   highlighted = False


   for span in doc.getElementsByType(Span):
       style_name = span.getAttribute('stylename')
       if style_name:
           for automatic_style in doc.automaticstyles.childNodes:
               if automatic_style.getAttribute('name') == style_name:
                   for property in automatic_style.childNodes:
                       if property.getAttribute('backgroundcolor') ==
'#ffff00':
                           highlighted = True
                           break
           if highlighted:
               break
```

```
    return 0 if highlighted else 1
```