OSBench Assessment Guideline — Overview

This document outlines the process for completing the OSBench assessment using the OSWorld-SFT framework. It guides you through setting up a reproducible environment on Windows, macOS, or Linux, understanding the task format, and generating the required submission artifacts.

You are expected to complete the workflow end to end: environment setup, repository exploration, task definition, and SFT trajectory creation.

Assessment Task

You must implement the following task in OSWorld-SFT as both a Task JSON and an SFT trajectory:

- Snapshot: `gimp`
- Instruction:
  Convert the image `photo.png` to black and white, then export the grayscale image to the Desktop as `gray_photo.png`

What This Guide Covers

The following sections explain how to:

- Clone and set up the OSWorld-SFT repository
  `git clone https://github.com/asad-a1-tr/OSWorld-SFT.git`
- Understand the OSWorld Task JSON schema
- Create a valid Task JSON for the given instruction
- Record and construct the corresponding SFT trajectory
- Validate the task execution within the VM environment

Detailed VM setup instructions, JSON schema definitions, and directory structures are provided later to reduce time spent on environment setup and codebase exploration.

Final Deliverables

Your submission must include:

1. Task JSON representing the provided instruction
2. SFT trajectory capturing the full task execution flow

Both artifacts must strictly follow the formats and structures described in the subsequent sections.

Local Installation - Environment Setup

1. Clone the Repository

```shell
git clone https://github.com/asad-a1-tr/OSWorld-SFT.git
cd OSWorld-SFT
```

---

2. Ensure Correct Python Version

- OSWorld-SFT requires **Python == 3.10**.
- System default may still point to an older version (e.g., 3.9).

Check installed versions:

```shell
python3 --version
```

Example output:

```none
Python 3.10.8
```

⚠️ Torch does **not** yet provide wheels for Python 3.13 on macOS ARM.
So we installed **Python 3.10** explicitly.

Install Python 3.10 with Homebrew (macOS):

```shell
brew install python@3.10
```

```
brew link python@3.10
```

Verify:

```Shell
python3.10 --version
# Python 3.10.x
```

---

3. Create a Virtual Environment with uv

We used **uv** (fast dependency resolver) to manage the venv:

```Shell
uv venv .venv --python=python3.10
source .venv/bin/activate
```

Check:

```Shell
python --version
# Python 3.10.x
```

---

4. Dependency Issues & Fixes

4.1 Torch

- Torch wheels are available for Python 3.10, so no manual fix is needed.

4.2 BORB Packaging Bug

- `borb==3.0.2` on PyPI is corrupted (duplicate file entries).

- Fix: install directly from GitHub.
- Replace borb in requirement.txt with the link below
    - Or install it manually:
        - uv pip install git+https://github.com/jorisschellekens/borb.git@master

```
None
git+https://github.com/jorisschellekens/borb.git@master
```

---

5. Updated Requirements

We cleaned up `requirements.txt` to loosen strict pins and add missing dependencies.
Final working set (excerpt):

```
None
numpy>=1.26.0
Pillow>=10.0.0
gymnasium>=0.29.0
requests>=2.31.0
torch>=2.2,<2.6
transformers>=4.38.0
...
git+https://github.com/jorisschellekens/borb.git@master
...
google-cloud-compute
google-cloud-storage
```

(Full list maintained in repo.)

---

6. Install Dependencies

From the repo root:

```shell
uv pip install -r requirements.txt
```

This installs:

- All standard dependencies
- Torch (compatible wheel for Python 3.10)
- Playwright (auto-installed via `setup.py` post-install hook)
- BORB directly from GitHub

---

7. Verify Installation

Run:

```shell
python -c "import desktop_env; print('OSWorld-SFT ready ✅')"
```

If successful, you'll see:

```none
OSWorld-SFT ready ✅
```

OSWorld-SFT Local Installation(VMware Setup)

1. Why VMware?

OSWorld-SFT tasks run inside a virtual desktop environment (Word, Excel, Notepad, browser, etc.).
To isolate the agent and ensure reproducibility, VMware is used. The scripts take care of the image download and configuration.

---

2. Installing VMware on macOS

1. **Sign up for Broadcom account**
    - VMware is now distributed by Broadcom.
    - Go to: Broadcom Support Portal
    - Create a free account and log in.
2. **Download VMware Fusion 13.6.2**
    - Direct link: VMware Fusion Free Downloads
    - Select **VMware Fusion Player 13.6.2 for macOS**.
    - Or
        i. Install it directly by accessing the DMG from here:
            1. https://drive.google.com/file/d/1lZupdvHR8pepLiNHXE90Ku-EcdPX89mv/view?usp=sharing
3. **Install**
    - Open the `.dmg` file after downloading.
    - Drag VMware Fusion to Applications.

- Open it → grant permissions (System Preferences → Security & Privacy → Allow).

4. **Verify**
    - Launch VMware Fusion.
    - You should see the “Create a New Virtual Machine” screen.

---

3. Installing VMware on Windows

1. **Download VMware Workstation 17.6.3**
    - Direct link:
      VMware Workstation 17.6.3 Installer *
2. **Install**
    - Run the `.exe` installer.
    - Follow on-screen instructions.
    - Reboot if prompted.

3. **Verify**
    - Open VMware Workstation.
    - You should see the “Home” screen with options to create a VM.

4 . Accept VMWare EULA:

It is important to have vmware started at least once before stepping forward, and allow downloading the missing modules, and accepting the EULA. You don't need to subscribe to the user experience program, and not to check for updates on start. Start vmware by:

```
Shell
        vmware
```

It is important to have the commands vmware and vmrun searchable and accessible from the prompt. If vmware is not accessible please make sure it is added to the system PATH and reboot.

5 . To download the ISO Image:

- Execute the command below after cloning the repo and installing the requirements.

```
Shell
        python Turing_tooling/run_manual.py --provider_name vmware
```

- This will download a file of around 11.3GB under the folder **vmware_vm_data**.
- If everything is fine, it will start the VM will start locally.

If you have done these steps, you can proceed with task creation.

Guide: Using Hugging Face for OSWorld-SFT File Hosting

1. Why Hugging Face?

- **Client requirement**: Earlier versions of OSWorld used Google Drive for task files (e.g., Word/Excel documents). Google Drive has download limits → if too many annotators or models fetch the same file, the link gets blocked.
- **Solution**: Host files on Hugging Face Datasets (or ModelScope) → stable, versioned, no download-blocking.
- **Integration**: OSWorld-SFT tasks can directly fetch from Hugging Face URLs in the `config` and `evaluator` fields.

Example usage in a task:

JSON

```
  "config": [
    {
      "type": "download",
      "parameters": {
        "files": [
          {
            "url":
"https://huggingface.co/datasets/Sample-User/Sample-repo/resolve/
main/Initial_file.png",
            "path": "/home/user/Desktop/photo.png"
          }
        ]
      }
    },
```

---

2. Create a Hugging Face Account

1. Go to https://huggingface.co/join.
2. Sign up with your gmail.
3. Verify your email address.
4. Once logged in, you'll see your profile page at:

```
None
https://huggingface.co/<your-username>
```

---

3. Create a Dataset Repository

1. Go to New Dataset.
2. Fill details:
    - **Owner**: your username or org (if you want a shared repo).
    - **Dataset name**: e.g., `osworld_tasks_files`.
    - **Visibility**: Public (required).
    - **License**: leave default.
3. Click **Create Dataset**.

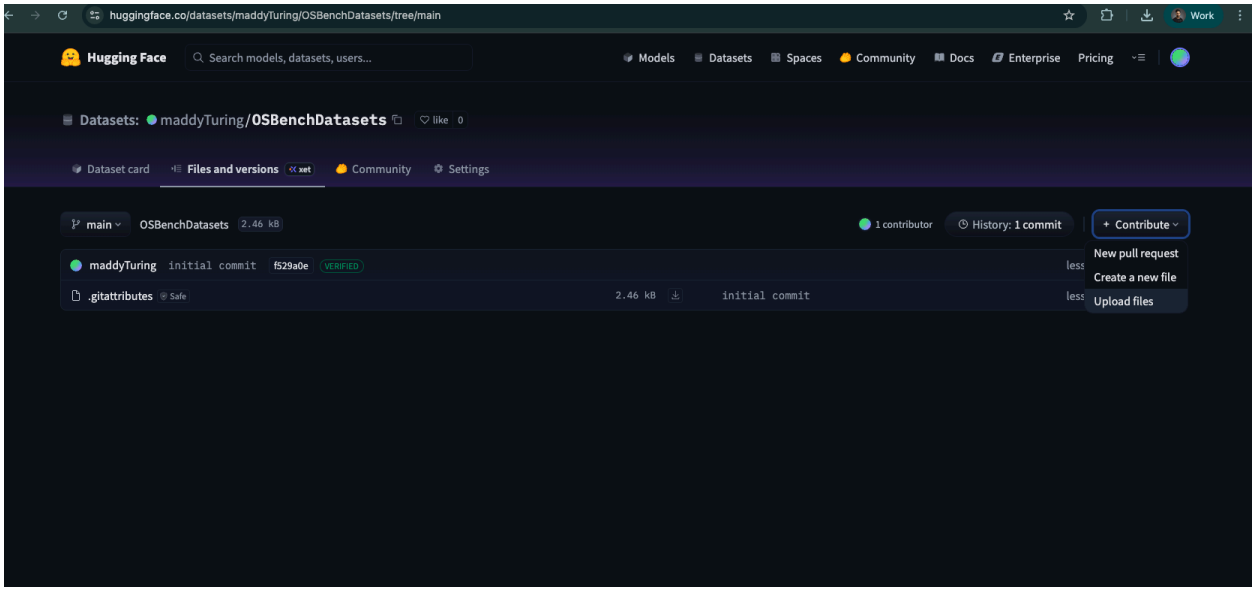
Your repo will be at:

```
None
https://huggingface.co/datasets/<your-username>/<your-repo>
```

---

4. Upload Files

Upload via Web UI



1. Go to your dataset repo → **Files and Versions**.
2. Click **Add file** → **Upload from computer**.
3. Drag & drop your `.docx`, `.xlsx`, `.pdf`, etc. (`.png` in your case)
4. Commit changes.

5. Get File Download Links

Each file in your dataset can be referenced directly via the **`resolve/main`** URL.

Example:
If you uploaded `budget.xlsx`, the link will be:

```
None
https://huggingface.co/datasets/<your-username>/<your-repo>/resolve/main/budget.xlsx
```

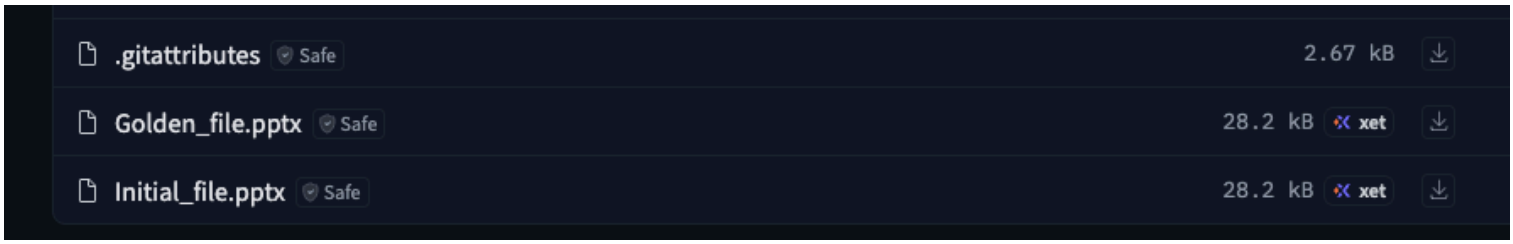
Use this in your task config as mentioned in the example usage above here

---

6. Best Practices

- Keep **initial files** (`Initial_file.png`) and **golden files** (`Golden_file.png`) in the same repo.
- Use clear filenames with suffixes:
    1. `Initial_file.png`
    2. `Golden_file.png`

Expected:



OSWorld-JSON file creation instructions(Phase - A)

The **JSON** file serves as a complete task blueprint that tells the system:

1) What files to download and where to put them in the VM (download config),
2) How to open the application with the file (launch config), and
3) What the LLM should accomplish (instruction field).

The system uses this JSON to automatically set up the VM environment, then sends the instruction to the LLM agent, which interacts with the VM through screenshots and actions to complete the task.

Key parameters:

config array: Automates VM setup (downloads, launches apps) before the AI starts
instruction: The actual task description sent to the LLM
evaluator: Defines how to check if the AI completed the task correctly
snapshot: Specifies which pre-configured VM environment to use (libreoffice_impress = Ubuntu with LibreOffice already installed)

Mandatory JSON File Contents (Failure to include these fields will result in disqualification from evaluation):

- **id:** Unique identifier for the task (UUID format)
- **snapshot:** VM environment to use (In your case: "libreoffice_impress" = Ubuntu with LibreOffice)
- **instruction:** Human-readable task description sent to the LLM (Defined already in the start)
- **source:** Reference URL or attribution for the task (Turing)
- **trajectory:** Directory path where execution logs are stored (Mostly it is: “trajectories/”)
- **config:** Array of setup steps (download, launch, etc.) before task starts
- **related_apps:** List of applications involved in the task
- **evaluator:** Defines how to check if the task was completed successfully(Choose it wisely from section **[OSWorld Evaluators Documentation]** as this is one of the reason for getting wrong evaluation score)
- **proxy:** Boolean indicating if proxy is needed for the task
- **fixed_ip:** Boolean indicating if VM needs a fixed IP address

- **possibility_of_env_change:** Risk level of environment modifications ("low", "medium", "high")
- **model_pass_rate:** Expected success rate for AI model (include `"claude-4-sonnet-20250514"` only)
- **annotator_hints:** Step-by-step guidance for human annotators
- **knowledge_points:** Key skills/concepts the task tests
- **coverage:** Brief description of what the task covers

Sample JSON FIle; Domain: VS Code

JSON


```
{
    "id": "a1b2c3d4-e5f6-7890-ab12-cdef34567890",
    "snapshot": "vscode",
    "instruction": "Open VS Code, open **User Settings
(JSON)** (not the read-only Default Settings) and set
\"editor.fontSize\" to 16 and write , after it. Then verify
the User settings.json contains \"\\\"editor.fontSize\\\":
16\".",
    "source": "custom_task_vscode_settings_check",
    "config": [
      {
        "type": "launch",
        "parameters": {
          "command": ["code"]
        }
      },
      {
        "type": "activate_window",
        "parameters": {
          "window_name": "Visual Studio Code"
        }
      }
    ],
    "trajectory": "trajectories/",
```

```
    "related_apps": ["vscode"],
    "evaluator": {
      "func": "is_extension_installed",
      "result": {
        "type": "vm_command_line",
        "command": [
          "sh",
          "-c",
          "cat ~/.config/Code/User/settings.json
2>/dev/null || cat ~/Library/Application\\
Support/Code/User/settings.json 2>/dev/null || echo '{}'"
        ]
      },
      "expected": {
        "type": "rule",
        "rules": {
          "type": "contain",
          "expected": "\"editor.fontSize\": 16"
        }
      }
    },
  "proxy": false,
  "fixed_ip": false,
  "possibility_of_env_change": "low",
  "model_pass_rate": {
    "claude-4-sonnet-20250514": 0.875
  }
  "annotator_hints": [
    "Step 1: Launch Visual Studio Code",
    "Step 2: Open User Settings (JSON)",
    "Step 3: Add or update the setting \"editor.fontSize\":
16",
    "Step 4: Save the changes and close settings.json"
```

```
  ],
  "knowledge_points": [
    "VS Code configuration",
    "JSON-based settings editing",
    "Verifying applied settings"
  ],
  "coverage": "Basic VS Code customization",

}
```

The evaluator function should be chosen correctly, or else it will affect and cause wrong evaluator result;

After creating the JSON file, place it in a new folder named "Deliverable" within the directory. Then, run the command below to validate the JSON file.

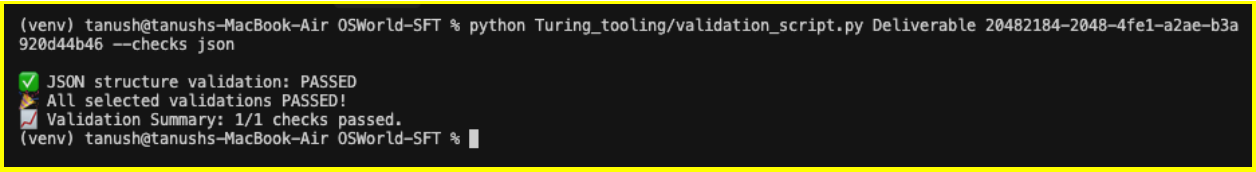
```
Shell
python Turing_tooling/validation_script.py Deliverable <task_id>
--checks json
```

This command is assuming file name of json file is <task_id>.json

You will see an output:

Based on the validation script code, here's what you'll see on successful JSON validation:

✅ JSON structure validation: PASSED



That's it! The validation script is designed to be concise - it only shows the success message when the JSON structure is valid according to the schema.On failure, you would see something like:

❌ JSON validation error: [specific error message about what's wrong]

Additional context:

- The script uses the `osworld_updated_schema.json` schema file to validate your JSON
- You can use this file to create JSON file too using LLM or manually.

On successful creation of JSON File we will move to next phase of SFT Creation

SFT Creation — Solution Generation

Objective

- **Create the best possible golden solution that completes the task exactly as expected.**
- **Record the exact sequence of GUI actions (e.g., clicks, keystrokes).**
- **Package the trajectory into the SFT folder alongside the task JSON.**

This manually curated demonstration will be used to fine-tune agents later.

🛠️ 6.1 Environment Setup

✅ Task File Placement:

Ensure your task JSON is stored under:

```
None
evaluation_examples/examples/<domain>/<task_id>.json
```

Example:

```
None
evaluation_examples/examples/vs_code/Turing2ba02ad5-d008-4f9e-a07f-729aae3c531b.json
```

🔁 VM Snapshot

- **Before you start, reset the VM to a clean snapshot (e.g., `init_state`) manually or through your VM provider.**
- **For retries, ALWAYS revert the VM before re-running to avoid side-effects.**

🚀 6.2 Run the Manual SFT Tool

Use the `run_manual.py` tool to step through the task and log each action.

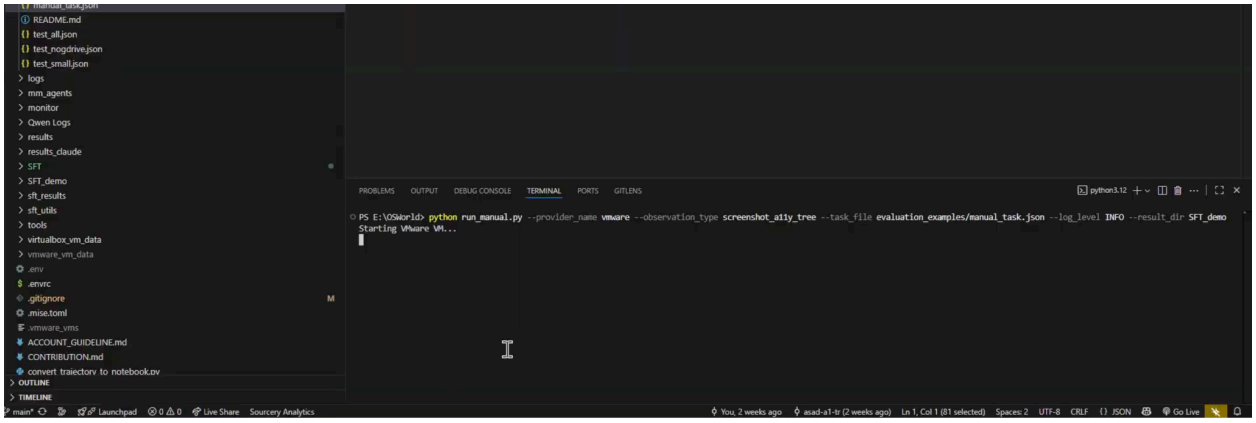
🔧 Command:

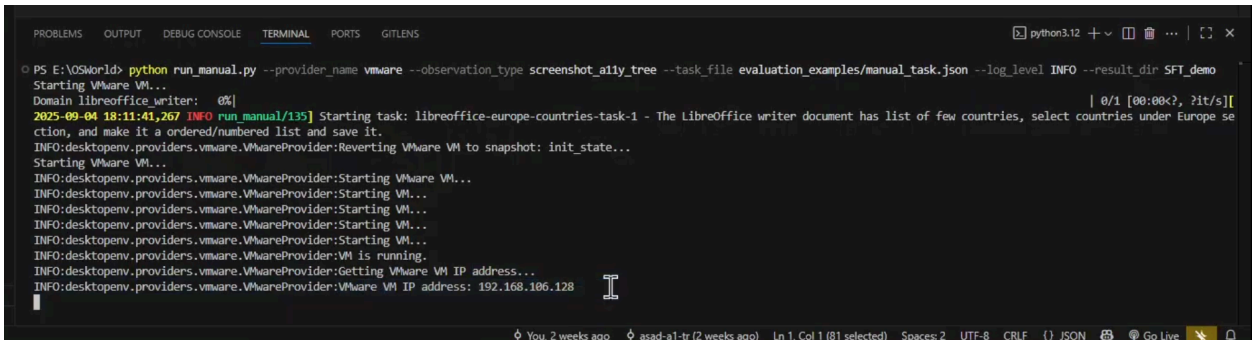
```
Shell

python Turing_tooling/run_manual.py --provider_name vmware
--observation_type screenshot_a11y_tree --task_file
evaluation_examples/manual_task.json --log_level INFO
--result_dir SFT
```

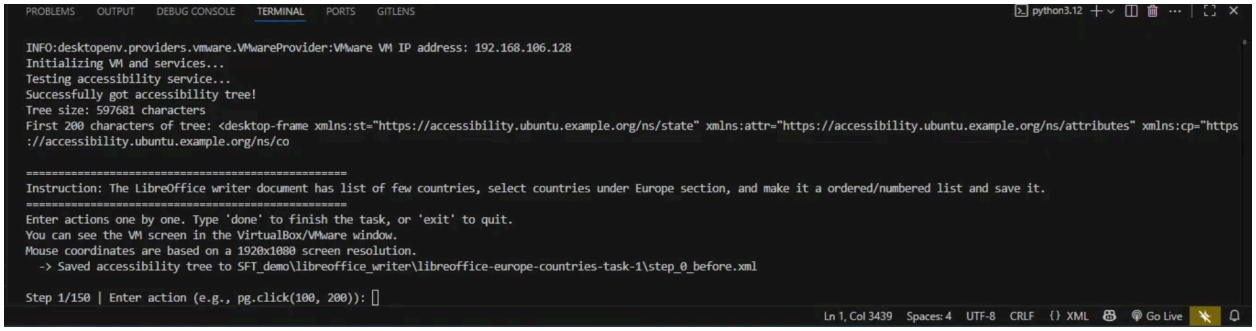
🔍 Flags Explained:

- **`--provider_name vmware`: Use VMware backend.**
- **`--observation_type screenshot_a11y_tree`: Collects both screenshot and accessibility tree per step.**
- **`--task_file`: Full path to the task JSON file.**
- **`--result_dir SFT`: Output folder for trajectories, screenshots, and logs.**

Running the above command will look something like this: Wait for the VM to start. It takes around 1-2 min to start(depends on your pc configuration.)







🧠 6.3 Developing the Golden Solution

You will be prompted with:

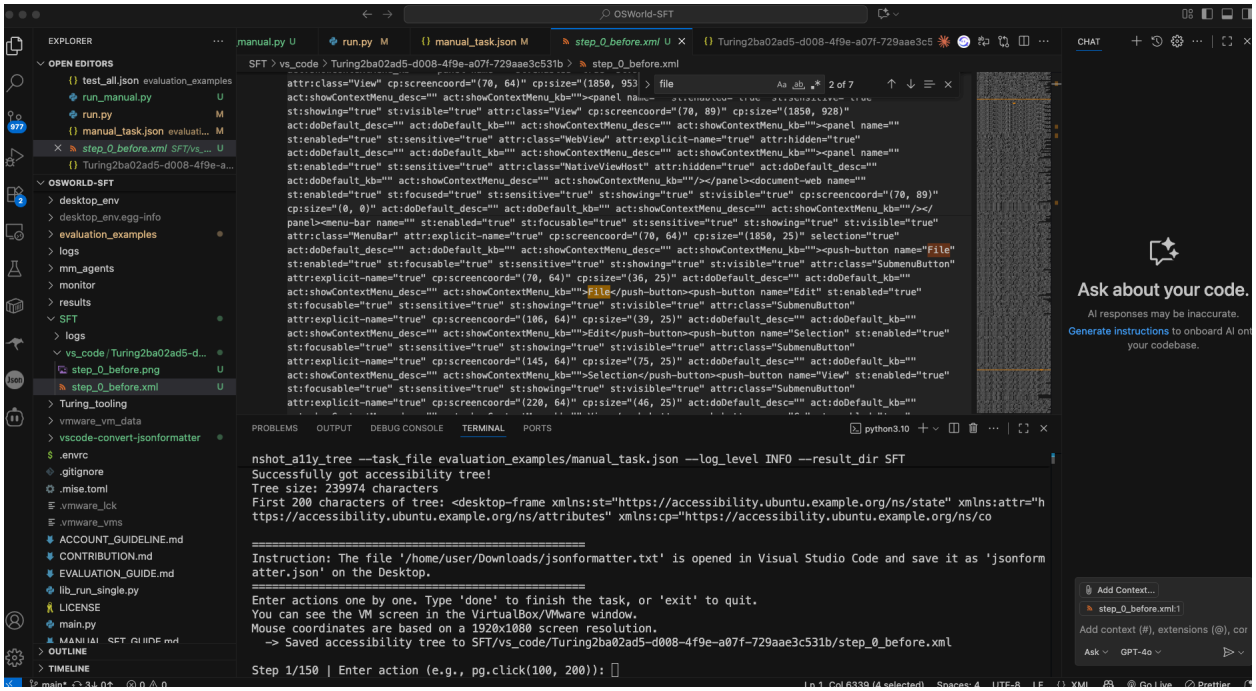
```shell
Enter actions one by one. Type 'done' to finish the task.
```

Each action you input is logged and executed in the VM.

✅ Use PyAutoGUI Commands

Refer to the [PyAutoGUI cheatsheet] for commands:

```python
pg.click(960, 540)

pg.hotkey("ctrl", "shift", "s")

pg.typewrite("jsonformatter.json")

pg.press("enter")
```



🖼️ 6.4 Understanding the State from Screenshot + XML

For each step, the tool generates:

- **A screenshot →** `step_0_before.png`
- **An accessibility XML tree →** `step_0_before.xml`

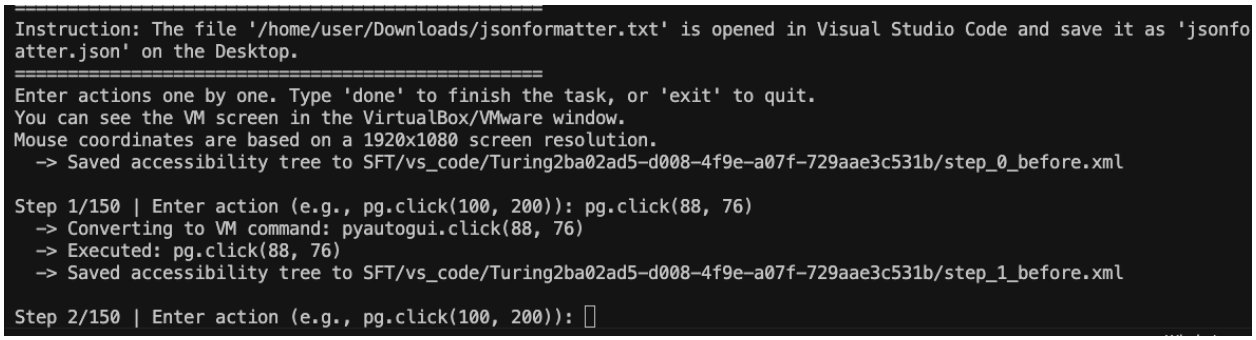
Use the XML file to find actionable elements:

Example:

```xml
cp:screencoord="(70, 64)"

cp:size="(36, 25)"
```

To click at the center:

```python
x = 70 + 36 / 2 = 88

y = 64 + 25 / 2 = 76

pg.click(88, 76)
```



❌ Common SFT Challenges + Workarounds

1. No Coordinates in XML

Ubuntu sidebar or floating elements may not return proper `screencoord` info.

✅ **Workaround:**

- **Use external coordinate tool:** https://www.programminghead.com/Projects/find-coordinates-of-image-online.html

- **Load screenshot and manually locate the approximate center.**
- **Use that as the click location.**

---

2. Stuck in Middle of Task?

- **Try a different valid path to solve the task (e.g., use Save As menu vs shortcut).**
- **Ask any LLM for creative but valid ideas.**

---

4. Validator Not Triggering

Make sure your evaluator block has the correct JSON keys.

✅ **Example:**

```JSON
"result": {
  "type": "vm_file",
  "path": "/home/user/Desktop/jsonformatter.json",
  "dest": "jsonformatter.json"
}
```

✅ 6.5 Finish and Save

Once you're done, type:

```None
done
```

The tool will:

- **Run the evaluator function.**
- **Write a `result.txt` (1.0 or 0.0)**
- **Save all actions into:**

```
SFT/<domain>/<task_id>/trajectory.jsonl

SFT/<domain>/<task_id>/step_*.xml

SFT/<domain>/<task_id>/step_*.png
```

You can now review the steps, actions, and success metrics.

📁 Folder Structure After SFT

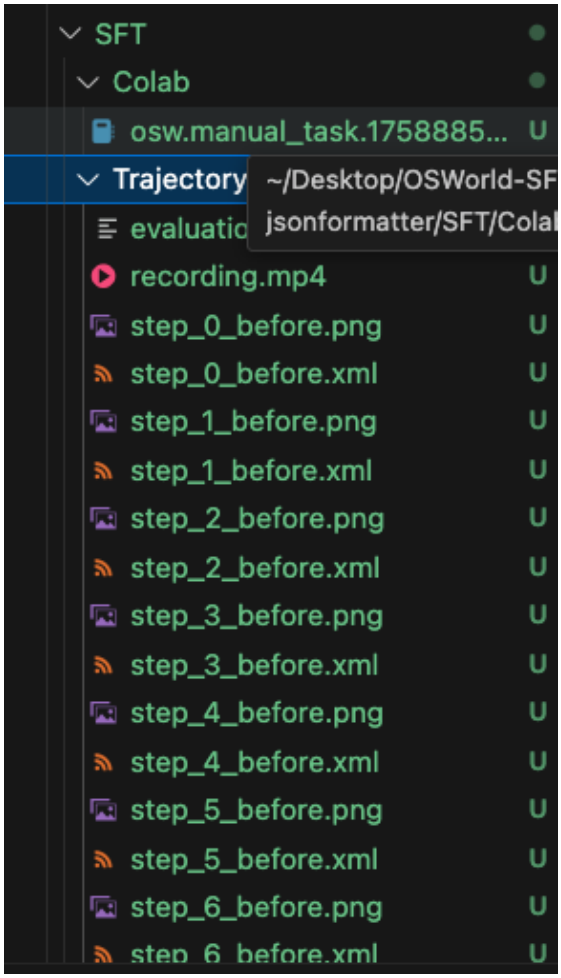
```
SFT/
└── vs_code/
    └── 2ba02ad5-d008-4f9e-a07f-729aae3c531b/
        ├── trajectory.jsonl
        ├── result.txt
        ├── step_0_before.xml
        ├── step_0_before.png
        ├── ...
```

📌 Final Tips

- **Use keyboard shortcuts for consistent cross-resolution support.**
- **Use `pg.hotkey("ctrl", "shift", "s")` for Save As.**
- **Use `pg.typewrite()` and `pg.press()` instead of clicking on text fields.**
- **Preview every screenshot carefully before each step.**
- **Retry if evaluation fails even after seemingly correct steps.**

Colab:

- The script will also produce a formatted colab with the logs.
- Collect and structure it in the deliverables:



- **The script provides only a skeleton Colab notebook. Ensure that you convert it to an Assistant or User conversation, where you need to write the thought process and response as if an LLM would generate them.**

Final Packaging & Submission

Once you have completed the entire task lifecycle (Phases A-D), it's time to finalize the task by properly organizing the directory, collecting all logs, and submitting the work.

📂 8.1 Folder Structure Overview

You should create a directory named:

```
None
<task_id>
```

Example: Let's say your uuid task id is: `e0eedf6d-9dc7-11f0-a98b-17892f803206.`
Directory name will look like this:

```
None
e0eedf6d-9dc7-11f0-a98b-17892f803206
```

Inside this directory, place the following content:

🔸 SFT/

Contains the **golden demonstration** trajectory

Structure:

```
None

e0eedf6d-9dc7-11f0-a98b-17892f803206
├──e0eedf6d-9dc7-11f0-a98b-17892f803206.json
├──SFT/
      ├── Colab/
```

```
|   └── osw.manual_task.1758818479.ipynb    (It will be
automatically created when you run SFT on your machine)
└── Trajectory and Screenshot/
    ├── trajectory.jsonl
    ├── evaluation_score.txt
    ├── recording.mp4  (Optional: if generated
automatically after running SFT)

    ├── manual-20250925@213313.log
    ├── step_N_before.png
    ├── step_N_before.xml
```

✅ This is your **best possible solution**, intended for SFT training.

✅ Make sure:

- Files are not zipped inside subfolders
- You've verified the presence of all result.txt and runtime.log
- You followed the **folder naming structure**

💡 Pro Tips

- Don't forget to **reset the VM snapshot** before each run (done automatically during SFT runs).
- Always **check for evaluator correctness** — invalid or missing result.txt = task rejection.
- Annotator trajectories are **critical** for final task acceptance. Build realistic mistakes!

🏁 That’s it!

You're done with all parts! ✅

You've now:

- Created a JSON
- Generated a golden SFT trajectory
- Added robust evaluators
- Properly packaged and submitted it 🎉

OSWorld Evaluators Documentation

This document provides a comprehensive overview of all available evaluators in the OSWorld project, including their usage, parameters, and examples.

Overview

Evaluators in OSWorld are functions that assess whether a task has been completed successfully by comparing the actual results with expected outcomes. They are organized into different modules based on the application or functionality they evaluate.

Evaluator Modules

1. General Evaluators (`general.py`)

These are general-purpose evaluators that can be used across different applications.

`exact_match(result, rules)`

- Purpose: Performs exact string matching
- Parameters:
  - `result`: The actual result string
  - `rules`: Dict with "expected" key containing the expected string
- Returns: 1.0 if exact match, 0.0 otherwise
- Example:

```JSON
{
  "func": "exact_match",
  "result": {"type": "vm_command_line", "command": ["echo",
"hello"]},
  "expected": {"expected": "hello"}
}
```

`fuzzy_match(result, rules)`

- Purpose: Performs fuzzy string matching using Levenshtein distance
- Parameters:
  - `result`: The actual result string
  - `rules`: Dict with "expected" key containing the expected string
- Returns: Similarity ratio (0.0 to 1.0)
- Example:

```JSON
{
  "func": "fuzzy_match",
  "result": {"type": "vm_command_line", "command": ["cat",
"file.txt"]},
  "expected": {"expected": "expected content"}
}
```

`check_include_exclude(result, rules)`

- Purpose: Checks if result contains required strings and excludes forbidden strings
- Parameters:
  - `result`: The result string to check
  - `rules`: Dict with "include" and "exclude" lists of strings
- Returns: 1.0 if all include strings are present and no exclude strings are present, 0.0 otherwise
- Example:

```JSON
{
  "func": "check_include_exclude",
  "result": {"type": "vm_file", "path":
"/home/user/output.txt"},
  "expected": {
    "include": ["success", "completed"],
    "exclude": ["error", "failed"]
  }
}
```

`check_csv(result, rules)`

- Purpose: Validates CSV file content against expected patterns
- Parameters:
  - `result`: Path to CSV file
  - `rules`: Dict with "expect" and "unexpect" lists of record patterns
- Returns: 1.0 if all expectations met, 0.0 otherwise
- Example:

```json
{
  "func": "check_csv",
  "result": {"type": "vm_file", "path":
"/home/user/data.csv"},
  "expected": {
    "expect": [{"name": "John", "age": "25"}],
    "unexpect": [{"status": "error"}]
  }
}
```

`check_json(result, rules)`

- Purpose: Validates JSON file content
- Parameters:
  - `result`: Path to JSON file
  - `rules`: Dict with "expect" and "unexpect" rules for JSON paths
- Returns: 1.0 if all validations pass, 0.0 otherwise
- Example:

```json
{
  "func": "check_json",
  "result": {"type": "vm_file", "path":
"/home/user/config.json"},
  "expected": {
    "expect": [{"key": ["database", "host"], "method":
"eq", "ref": "localhost"}],
```

```
    "unexpect": [{"key": ["error"], "method": "exists"}]
  }
}
```

`check_accessibility_tree(result, rules, osname)`

- Purpose: Validates accessibility tree structure
- Parameters:
  - `result`: XML string of accessibility tree
  - `rules`: List of validation rules with selectors and expected text
  - `osname`: OS name ("ubuntu", "windows", "macos")
- Returns: Similarity score (0.0 to 1.0)
- Example:

```JSON
{
  "func": "check_accessibility_tree",
  "result": {"type": "accessibility_tree"},
  "expected": [
    {
      "selectors": ["button"],
      "text": "Save",
      "exact": true
    }
  ]
}
```

`file_contains(file_path, config)`

- Purpose: Checks if file contains specified strings
- Parameters:
  - `file_path`: Path to file to check
  - `config`: Dict with "expected" list of strings that must be present
- Returns: 1.0 if all strings found, 0.0 otherwise

`compare_terminal_and_txt(txt_file_path, terminal_output)`

- Purpose: Compares terminal output with expected text file
- Parameters:
  - `txt_file_path`: Path to expected text file
  - `terminal_output`: Actual terminal output string
- Returns: 1.0 if match, 0.0 otherwise

`compare_python_pure_text(py_file_path, gold_file_path)`

- Purpose: Compares Python files ignoring metadata and formatting
- Parameters:
  - `py_file_path`: Path to actual Python file
  - `gold_file_path`: Path to expected Python file
- Returns: 1.0 if functionally equivalent, 0.0 otherwise

2. Table Evaluators (`table.py`)

`compare_table(result, expected, **options)`

- Purpose: Comprehensive Excel/CSV file comparison
- Parameters:
  - `result`: Path to actual file
  - `expected`: Path to expected file (optional)
  - `options`: Dict with "rules" list defining comparison rules
- Supported Rule Types:
  - `sheet_name`: Compare sheet names
  - `sheet_data`: Compare internal cell values
  - `sheet_print`: Compare displayed cell values
  - `sheet_fuzzy`: Fuzzy matching with ranges
  - `sparkline`: Compare sparklines
  - `chart`: Compare charts
  - `style`: Compare cell styles
  - `freeze`: Compare freeze panes
  - `zoom`: Check zoom level
  - `data_validation`: Compare data validation rules
  - `row_props`: Compare row properties
  - `col_props`: Compare column properties
  - `filter`: Compare filters
  - `pivot_table`: Compare pivot tables
  - `check_cell`: Check specific cell properties
- Returns: 1.0 if all rules pass, 0.0 otherwise
- Example:

```JSON
{
  "func": "compare_table",
  "result": {"type": "vm_file", "path":
"/home/user/result.xlsx"},
  "expected": {"type": "cloud_file", "path":
"expected.xlsx"},
  "options": {
    "rules": [
      {
        "type": "sheet_data",
        "sheet_idx0": 0,
        "sheet_idx1": "EI0"
      }
    ]
  }
}
```

`compare_csv(result, expected, **options)`

- Purpose: Compare CSV files
- Parameters: Similar to compare_table but for CSV files
- Returns: 1.0 if match, 0.0 otherwise

`compare_conference_city_in_order(actual_city_list_path, expected_city)`

- Purpose: Validates conference cities are in correct order
- Parameters:
  - `actual_city_list_path`: Path to Excel file with cities
  - `expected_city`: Dict with expected city order
- Returns: 1.0 if correct order, 0.0 otherwise

3. Chrome Evaluators (`chrome.py`)

`is_expected_tabs(open_tabs, rule)`

- Purpose: Validates open tabs in Chrome
- Parameters:

    - `open_tabs`: List of open tab dictionaries
    - `rule`: Dict with "type" and matching criteria
- Supported Types:
    - `url`: Match tab URLs
- Returns: 1.0 if tabs match expectation, 0.0 otherwise
- Example:

```JSON
{
  "func": "is_expected_tabs",
  "result": {"type": "open_tabs"},
  "expected": {
    "type": "url",
    "urls": ["https://example.com", "https://google.com"]
  }
}
```

`is_expected_active_tab(active_tab_info, rule)`

- Purpose: Validates the currently active tab
- Parameters:
    - `active_tab_info`: Dict with active tab information
    - `rule`: Dict with matching criteria
- Returns: 1.0 if active tab matches, 0.0 otherwise

`is_expected_bookmarks(bookmarks, rule)`

- Purpose: Validates Chrome bookmarks
- Parameters:
    - `bookmarks`: Bookmarks structure
    - `rule`: Dict with bookmark validation rules
- Supported Types:
    - `bookmark_bar_folders_names`
    - `bookmark_bar_websites_urls`
    - `liked_authors_websites_urls`
- Returns: 1.0 if bookmarks match, 0.0 otherwise

`compare_pdfs(pdf1_path, pdf2_path)`

- Purpose: Compare PDF files by text content
- Parameters:

    - `pdf1_path`: Path to first PDF
    - `pdf2_path`: Path to second PDF
- Returns: Similarity score (0.0 to 1.0)

`compare_htmls(html_path1, html_path2, **options)`

- Purpose: Compare HTML files
- Parameters:
    - `html_path1`: Path to first HTML file
    - `html_path2`: Path to second HTML file
    - `options`: Dict with comparison options
- Returns: 1.0 if identical, 0.0 otherwise

`is_cookie_deleted(cookie_data, rule)`

- Purpose: Checks if cookies are deleted
- Parameters:
    - `cookie_data`: List of cookie data
    - `rule`: Dict with deletion criteria
- Returns: 1.0 if cookies deleted, 0.0 otherwise

`is_shortcut_on_desktop(shortcuts, rule)`

- Purpose: Checks if shortcut exists on desktop
- Parameters:
    - `shortcuts`: Dict of desktop shortcuts
    - `rule`: Dict with shortcut criteria
- Returns: 1.0 if shortcut found, 0.0 otherwise

4. VS Code Evaluators (`vscode.py`)

`check_json_keybindings(actual, expected)`

- Purpose: Validates VS Code keybindings
- Parameters:
    - `actual`: Path to keybindings file
    - `expected`: Dict with expected keybinding
- Returns: 1.0 if keybinding found, 0.0 otherwise

`check_json_settings(actual, expected)`

- Purpose: Validates VS Code settings
- Parameters:
    - `actual`: Path to settings file
    - `expected`: Dict with expected settings

- Returns: 1.0 if settings match, 0.0 otherwise

`compare_text_file(actual, expected, **options)`

- Purpose: Compare text files with various options
- Parameters:
  - `actual`: Path to actual file
  - `expected`: Path to expected file
  - `options`: Dict with comparison options (ignore_blanks, ignore_case)
- Returns: 1.0 if match, 0.0 otherwise

check_json_settings(actual, expected)

- **Purpose:** Validates VS Code settings JSON.
- **Parameters:**
  - `actual`: path to the settings JSON file
  - `expected`: dict with `"expected"` key/value pairs to check
- **Returns:** `1.0` if all expected settings match, else `0.0`.

`is_extension_installed(actual, rules)`

- Purpose: Checks if VS Code extension is installed
- Parameters:
  - `actual`: Output of `code --list-extensions`
  - `rules`: Dict with check type and expected extension
- Returns: 1.0 if extension installed, 0.0 otherwise
- Example:

```json
{
  "func": "is_extension_installed",
  "result": {"type": "vm_command_line", "command": ["code",
"--list-extensions"]},
```

```
  "expected": {"type": "contain", "expected":
"ms-python.python"}
}
```

`check_python_file_by_test_suite(actual_files, test_file, **options)`

- Purpose: Runs test suite on Python files
- Parameters:
  - `actual_files`: List of Python files to test
  - `test_file`: Path to test file
  - `options`: Dict with test options
- Returns: Test result (0.0 to 1.0)

check_python_file_by_gold_file(actual_files, gold_file, **options)

- **Purpose:** (Stub) Intended to compare Python files against a gold reference.
- **Status:** Not yet implemented.

5. Document Evaluators (`docs.py`)

`compare_docx_files(file1, file2, **options)`

- Purpose: Compare Word documents
- Parameters:
  - `file1`: Path to first document
  - `file2`: Path to second document
  - `options`: Dict with comparison options
- Supported Options:
  - `ignore_blanks`: Ignore whitespace differences
  - `ignore_case`: Case-insensitive comparison
  - `ignore_order`: Ignore paragraph order
  - `content_only`: Compare content only
  - `fuzzy_match`: Use fuzzy matching

    - `delete_empty_lines`: Remove empty lines
- Returns: Similarity score (0.0 to 1.0)

`contains_page_break(docx_file, rules)`

- Purpose: Checks for page breaks in document
- Parameters:
    - `docx_file`: Path to Word document
    - `rules`: Dict with expected page break count
- Returns: 1.0 if page breaks found, 0.0 otherwise

`find_default_font(config_file_path, rules)`

- Purpose: Finds default font in LibreOffice config
- Parameters:
    - `config_file_path`: Path to LibreOffice config
    - `rules`: Dict with expected font name
- Returns: 1.0 if font matches, 0.0 otherwise

6. GIMP Evaluators (`gimp.py`)

`compare_image_list(pred_img_path_list, gold_img_path_list)`

- Purpose: Compare lists of images for exact pixel match
- Parameters:
    - `pred_img_path_list`: List of predicted image paths
    - `gold_img_path_list`: List of expected image paths
- Returns: 1.0 if all images match exactly, 0.0 otherwise

`check_file_exists(directory, filename)`

- Purpose: Check if file exists in directory
- Parameters:
    - `directory`: Directory path
    - `filename`: Filename to check
- Returns: 1.0 if exists, 0.0 otherwise

`increase_saturation(image1_path, image2_path)`

- Purpose: Check if saturation increased from image1 to image2
- Parameters:
    - `image1_path`: Original image path
    - `image2_path`: Modified image path
- Returns: 1.0 if saturation increased, 0.0 otherwise

`decrease_brightness(image1_path, image2_path)`

- Purpose: Check if brightness decreased from image1 to image2
- Parameters:
  - `image1_path`: Original image path
  - `image2_path`: Modified image path
- Returns: 1.0 if brightness decreased, 0.0 otherwise

`compare_triangle_positions(image1, image2)`

- Purpose: Compare positions of yellow triangles in images
- Parameters:
  - `image1`: First image path
  - `image2`: Second image path
- Returns: 1.0 if triangle moved closer to center, 0.0 otherwise

`check_brightness_decrease_and_structure_sim(src_path, tgt_path)`

- Purpose: Check brightness decreased and structure is similar
- Parameters:
  - `src_path`: Source image path
  - `tgt_path`: Target image path
- Returns: 1.0 if brightness decreased and structure similar, 0.0 otherwise

`check_saturation_increase_and_structure_sim(src_path, tgt_path)`

- Purpose: Check saturation increased and structure is similar
- Parameters:
  - `src_path`: Source image path
  - `tgt_path`: Target image path
- Returns: 1.0 if saturation increased and structure similar, 0.0 otherwise

`check_file_exists_and_structure_sim(src_path, tgt_path)`

- Purpose: Check file exists and structure is similar to target
- Parameters:
  - `src_path`: Source image path
  - `tgt_path`: Target image path
- Returns: 1.0 if file exists and structure similar, 0.0 otherwise

`check_triangle_position(tgt_path)`

- Purpose: Check if triangle is in center of image
- Parameters:
  - `tgt_path`: Target image path
- Returns: 1.0 if triangle is centered, 0.0 otherwise

`check_structure_sim(src_path, tgt_path)`

- Purpose: Check if image structures are similar
- Parameters:
  - `src_path`: Source image path
  - `tgt_path`: Target image path
- Returns: 1.0 if structures similar, 0.0 otherwise

`check_structure_sim_resized(src_path, tgt_path)`

- Purpose: Check structure similarity after resizing
- Parameters:
  - `src_path`: Source image path
  - `tgt_path`: Target image path
- Returns: 1.0 if structures similar after resize, 0.0 otherwise

`check_contrast_increase_and_structure_sim(src_path, tgt_path)`

- Purpose: Check contrast increased and structure is similar
- Parameters:
  - `src_path`: Source image path
  - `tgt_path`: Target image path
- Returns: 1.0 if contrast increased and structure similar, 0.0 otherwise

`check_config_status(actual_config_path, rule)`

- Purpose: Check GIMP configuration status
- Parameters:
  - `actual_config_path`: Config file path
  - `rule`: Dict with key and expected value
- Returns: 1.0 if config matches, 0.0 otherwise

`check_image_size(src_path, rule)`

- Purpose: Check image dimensions
- Parameters:
  - `src_path`: Image path
  - `rule`: Dict with width/height requirements
- Returns: 1.0 if size matches, 0.0 otherwise

`check_palette_and_structure_sim(src_path, tgt_path)`

- Purpose: Check if image is palette-based and structure similar
- Parameters:
  - `src_path`: Source image path
  - `tgt_path`: Target image path

- Returns: 1.0 if palette-based and structure similar, 0.0 otherwise

`check_textbox_on_leftside(src_path)`

- Purpose: Check if textbox is on left side of image
- Parameters:
  - `src_path`: Image path
- Returns: 1.0 if textbox on left, 0.0 otherwise

`check_image_mirror(src_path, tgt_path)`

- Purpose: Check if image is mirrored
- Parameters:
  - `src_path`: Source image path
  - `tgt_path`: Target image path
- Returns: 1.0 if mirrored, 0.0 otherwise

`check_green_background(src_path, tgt_path)`

- Purpose: Check if background is green
- Parameters:
  - `src_path`: Source image path
  - `tgt_path`: Target image path
- Returns: 1.0 if background is green, 0.0 otherwise

`check_sharper(src_path, tgt_path)`

- Purpose: Check if source image is sharper than target
- Parameters:
  - `src_path`: Source image path
  - `tgt_path`: Target image path
- Returns: 1.0 if sharper, 0.0 otherwise

`check_image_file_size(src_path, rule)`

- Purpose: Check image file size
- Parameters:
  - `src_path`: Image path
  - `rule`: Dict with max_size requirement
- Returns: 1.0 if size within limit, 0.0 otherwise

7. VLC Evaluators (`vlc.py`)

`is_vlc_playing(actual_status_path, rule)`

- Purpose: Check if VLC is currently playing and what file/URL

- Parameters:
  - `actual_status_path`: Path to VLC status XML file
  - `rule`: Dict with "type" ("file_name" or "url") and expected value
- Returns: 1.0 if playing expected content, 0.0 otherwise

`is_vlc_recordings_folder(actual_config_path, rule)`

- Purpose: Check VLC's recording folder setting
- Parameters:
  - `actual_config_path`: Path to VLC config file
  - `rule`: Dict with "recording_file_path"
- Returns: 1.0 if recording path matches, 0.0 otherwise

`is_vlc_fullscreen(actual_window_size, screen_size)`

- Purpose: Check if VLC window is fullscreen
- Parameters:
  - `actual_window_size`: Current window size
  - `screen_size`: Screen size
- Returns: 1.0 if fullscreen, 0.0 otherwise

`compare_images(image1_path, image2_path, **options)`

- Purpose: Compare images using SSIM (Structural Similarity Index)
- Parameters:
  - `image1_path`: Path to first image
  - `image2_path`: Path to second image
  - `options`: Dict with comparison options
- Returns: Similarity score (0.0 to 1.0)

`compare_audios(audio_path_1, audio_path_2)`

- Purpose: Compare audio files using MFCC and DTW
- Parameters:
  - `audio_path_1`: Path to first audio file
  - `audio_path_2`: Path to second audio file
- Returns: Similarity score (0.0 to 1.0)

`compare_videos(video_path1, video_path2, **options)`

- Purpose: Compare videos by sampling frames and comparing hashes
- Parameters:
  - `video_path1`: Path to first video
  - `video_path2`: Path to second video
  - `options`: Dict with max_frames_to_check and threshold

- Returns: 1.0 if videos match, 0.0 otherwise

`check_qt_bgcone(actual_config_path, rule)`

- Purpose: Check VLC's qt-bgcone setting
- Parameters:
  - `actual_config_path`: Path to VLC config file
  - `rule`: Dict with "expected_qt_bgcone"
- Returns: 1.0 if setting matches, 0.0 otherwise

`check_qt_max_volume(actual_config_path, rule)`

- Purpose: Check VLC's maximum volume setting
- Parameters:
  - `actual_config_path`: Path to VLC config file
  - `rule`: Dict with "expected_qt_max_volume"
- Returns: 1.0 if setting matches, 0.0 otherwise

`check_qt_minimal_view(actual_config_path, rule)`

- Purpose: Check VLC's minimal view setting
- Parameters:
  - `actual_config_path`: Path to VLC config file
  - `rule`: Dict with "expected_qt_minimal_view"
- Returns: 1.0 if setting matches, 0.0 otherwise

`check_qt_slider_colours(actual_config_path, rule)`

- Purpose: Check VLC's slider colors
- Parameters:
  - `actual_config_path`: Path to VLC config file
  - `rule`: Dict with color checking rules
- Returns: 1.0 if colors match criteria, 0.0 otherwise

`check_global_key_play_pause(actual_config_path, rule)`

- Purpose: Check VLC's global play/pause key setting
- Parameters:
  - `actual_config_path`: Path to VLC config file
  - `rule`: Dict with "expected_global_key_play_pause"
- Returns: 1.0 if setting matches, 0.0 otherwise

`check_one_instance_when_started_from_file(actual_config_path, rule)`

- Purpose: Check VLC's single instance setting

- Parameters:
  - `actual_config_path`: Path to VLC config file
  - `rule`: Dict with "expected_one_instance_when_started_from_file"
- Returns: 1.0 if setting matches, 0.0 otherwise

`check_libre_locale(config_file, rules)`

- Purpose: Check LibreOffice locale settings
- Parameters:
  - `config_file`: Path to LibreOffice config file
  - `rules`: Dict with "locale_set" list of expected locales
- Returns: 1.0 if locale matches any in set, 0.0 otherwise

8. Slides Evaluators (`slides.py`)

`compare_pptx_files(file1, file2, **options)`

- Purpose: Compare PowerPoint files
- Parameters:
  - `file1`: Path to first presentation
  - `file2`: Path to second presentation
  - `options`: Dict with comparison options
- Returns: Similarity score (0.0 to 1.0)

`check_slide_numbers_color(pptx_file, expected_color)`

- Purpose: Check slide number color
- Parameters:
  - `pptx_file`: Path to PowerPoint file
  - `expected_color`: Expected color
- Returns: 1.0 if color matches, 0.0 otherwise

`check_presenter_console_disable(config_file_path)`

- Purpose: Check if presenter console is disabled in LibreOffice Impress
- Parameters:
  - `config_file_path`: Path to LibreOffice config file
- Returns: 1.0 if disabled, 0.0 otherwise

`check_image_stretch_and_center(modified_ppt, original_ppt)`

- Purpose: Check if image is stretched and centered on slide
- Parameters:
  - `modified_ppt`: Path to modified presentation
  - `original_ppt`: Path to original presentation

- Returns: 1.0 if image is properly stretched and centered, 0.0 otherwise

9. Thunderbird Evaluators (`thunderbird.py`)

`check_thunderbird_prefs(result, rule)`

- Purpose: Check Thunderbird preferences
- Parameters:
  - `result`: Path to Thunderbird prefs file
  - `rule`: Dict with "expect" and "unexpect" preference rules
- Returns: 1.0 if preferences match rules, 0.0 otherwise

`check_thunderbird_filter(result, rules)`

- Purpose: Check Thunderbird message filters
- Parameters:
  - `result`: Path to filter definition file
  - `rules`: Dict with "expect" and "unexpect" filter rules
- Returns: 1.0 if filters match rules, 0.0 otherwise

`check_thunderbird_folder(result, reference, **kwargs)`

- Purpose: Compare Thunderbird folder contents
- Parameters:
  - `result`: Path(s) to result folder file(s)
  - `reference`: Path(s) to reference folder file(s)
  - `kwargs`: Options for comparison (ignore_status, ignore_keys, etc.)
- Returns: 1.0 if folders match, 0.0 otherwise

10. PDF Evaluators (`pdf.py`)

`check_pdf_pages(pdf_file, rules)`

- Purpose: Check number of pages in PDF
- Parameters:
  - `pdf_file`: Path to PDF file
  - `rules`: Dict with "relation" (e.g., "eq", "gt") and "ref_value"
- Returns: 1.0 if page count matches rule, 0.0 otherwise

`extract_answers_from_pdf(pdf_file)`

- Purpose: Extract answers from PDF (utility function)
- Parameters:
  - `pdf_file`: Path to PDF file

- Returns: List of extracted answers

11. Basic OS Evaluators (`basic_os.py`)

`check_gnome_favorite_apps(apps_str, rule)`

- Purpose: Check GNOME favorite apps
- Parameters:
  - `apps_str`: String representation of apps list
  - `rule`: Dict with "expected" list of expected favorite apps
- Returns: 1.0 if apps match, 0.0 otherwise

`is_utc_0(timedatectl_output)`

- Purpose: Check if system timezone is UTC
- Parameters:
  - `timedatectl_output`: Output of timedatectl command
- Returns: 1.0 if UTC (+0000), 0.0 otherwise

`check_text_enlarged(scaling_factor_str)`

- Purpose: Check if text scaling factor is greater than 1.0
- Parameters:
  - `scaling_factor_str`: String representation of scaling factor
- Returns: 1.0 if enlarged, 0.0 otherwise

`check_moved_jpgs(directory_list, rule)`

- Purpose: Check if JPG files have been moved to directory
- Parameters:
  - `directory_list`: Directory listing structure
  - `rule`: Dict with "expected" list of expected JPG files
- Returns: 1.0 if JPGs moved correctly, 0.0 otherwise

`is_in_vm_clickboard(config, terminal_output)`

- Purpose: Check if expected content is in VM clipboard
- Parameters:
  - `config`: Dict with "expected" content
  - `terminal_output`: Terminal output to check
- Returns: 1.0 if content found, 0.0 otherwise

12. Others Evaluators (`others.py`)

`compare_epub(result, expected)`

- Purpose: Compare EPUB files by extracting and comparing contents
- Parameters:
  - `result`: Path to result EPUB file
  - `expected`: Path to expected EPUB file
- Returns: Average similarity score across all files (0.0 to 1.0)

`check_mp3_meta(result, meta)`

- Purpose: Check MP3 metadata tags
- Parameters:
  - `result`: Path to MP3 file
  - `meta`: Dict with metadata field rules
- Returns: 1.0 if all metadata matches rules, 0.0 otherwise

Special Evaluators

`infeasible()`

- Purpose: Marks a task as infeasible (cannot be completed)
- Parameters: None
- Returns: None (always considered failure)
- Example:

```json
{
  "func": "infeasible"
}
```

Usage in JSON Configuration

Evaluators are specified in the `evaluator` field of task JSON files:

```json
{
  "evaluator": {
    "func": "exact_match",
    "result": {
```

```
      "type": "vm_command_line",
      "command": ["echo", "hello world"]
    },
    "expected": {
      "expected": "hello world"
    }
  }
}
```

For more complex evaluators like `compare_table`:

JSON

```
{
  "evaluator": {
    "func": "compare_table",
    "result": {
      "type": "vm_file",
      "path": "/home/user/result.xlsx"
    },
    "expected": {
      "type": "cloud_file",
      "path": "expected.xlsx",
      "dest": "expected.xlsx"
    },
    "options": {
      "rules": [
        {
          "type": "sheet_data",
          "sheet_idx0": 0,
          "sheet_idx1": "EI0"
        }
      ]
    }
  }
}
```

Result Types

The `result` field can have different types:

- `vm_file`: File on the virtual machine
- `vm_command_line`: Output of a command
- `cloud_file`: File from cloud storage
- `accessibility_tree`: Accessibility tree XML
- `open_tabs`: Chrome open tabs
- `active_tab`: Chrome active tab info

Expected Types

The `expected` field can have different types:

- `rule`: Rule-based expectations
- `cloud_file`: Expected file from cloud
- `vm_file`: Expected file on VM

Notes

- All evaluators return a float between 0.0 and 1.0, where 1.0 indicates success
- Some evaluators support additional options for customization
- File paths should be absolute when specified
- Some evaluators require specific libraries to be installed (see README.md for details)

🖱️ Mouse Controls (pg)

```
# --- Movement ---
pg.moveTo(x, y, duration=1)        # Move to absolute screen position
pg.moveRel(dx, dy, duration=1)     # Move relative to current position
pg.position()                      # Get current mouse coordinates
pg.size()                          # Get screen resolution

# --- Clicking ---
pg.click(x, y)                     # Left click (optionally at x,y)
pg.doubleClick(x, y)               # Double left click
pg.rightClick(x, y)                # Right click
pg.middleClick(x, y)               # Middle click
pg.tripleClick(x, y)               # Triple click

# --- Press & Release ---
pg.mouseDown(x, y, button='left')  # Press (hold down) mouse button
pg.mouseUp(x, y, button='left')    # Release mouse button
pg.click(button='right')           # Explicit button argument

# --- Dragging ---
pg.dragTo(x, y, duration=1, button='left')   # Drag to absolute coords
pg.dragRel(dx, dy, duration=1, button='left')# Drag relative to pos
pg.drag(xOffset, yOffset, duration=1)        # Alias for dragRel()

# --- Scrolling ---
pg.scroll(500)                     # Vertical scroll (+up / -down)
pg.vscroll(-200)                   # Vertical scroll (explicit)
pg.hscroll(200)                    # Horizontal scroll
```

⌨️ Keyboard Controls (pg)

```
# --- Typing ---
pg.write("Hello World", interval=0.1)   # Type text with delay
```

```
pg.typewrite("Hello")                    # Alias for write()

# --- Single Key Press ---
pg.press("enter")                        # Press one key
pg.press(["left", "backspace", "enter"])# Press sequence of keys

# --- Hold & Release ---
pg.keyDown("shift")                      # Hold down a key
pg.keyUp("shift")                        # Release key

# --- Combos ---
pg.hotkey("ctrl", "c")                   # Combo (copy)
pg.hotkey("ctrl", "shift", "esc")        # Multi-key combo

# --- Arrow Keys ---
pg.press("up")                           # Up arrow
pg.press("down")                         # Down arrow
pg.press("left")                         # Left arrow
pg.press("right")                        # Right arrow

# --- Navigation Keys ---
pg.press("tab")                          # Tab
pg.press("esc")                          # Escape
pg.press("home")                         # Home
pg.press("end")                          # End
pg.press("pageup")                       # Page Up
pg.press("pagedown")                     # Page Down

# --- Editing Keys ---
pg.press("backspace")                    # Backspace
pg.press("delete")                       # Delete
pg.press("insert")                       # Insert
pg.press("space")                        # Spacebar
pg.press("enter")                        # Enter

# --- Function Keys ---
pg.press("f1")                           # F1
pg.press("f5")                           # Refresh
```

```
pg.press("f12")                         # Dev tools (browser)

# --- Modifier Keys ---
pg.press("shift")                       # Shift
pg.press("ctrl")                        # Ctrl
pg.press("alt")                         # Alt
pg.press("win")                         # Windows key
pg.press("command")                     # Command (macOS)

# --- Lock Keys ---
pg.press("capslock")                    # Caps Lock
pg.press("numlock")                     # Num Lock
pg.press("scrolllock")                  # Scroll Lock
```