AutoGenBench

AutoGenBench (agbench) is a tool for repeatedly running a set of pre-defined AutoGen tasks in a setting with tightly-controlled initial conditions. With each run, AutoGenBench will start from a blank slate. The agents being evaluated will need to work out what code needs to be written, and what libraries or dependencies to install, to solve tasks. The results of each run are logged, and can be ingested by analysis or metrics scripts (such as `agbench tabulate`). By default, all runs are conducted in freshly-initialized docker containers, providing the recommended level of consistency and safety.

AutoGenBench works with all AutoGen 0.1*., and 0.2.* versions.

Technical Specifications

If you are already an AutoGenBench pro, and want the full technical specifications, please review the contributor's guide.

Docker Requirement

AutoGenBench also requires Docker (Desktop or Engine). **It will not run in GitHub codespaces**, unless you opt for native execution (which is strongly discouraged). To install Docker Desktop see https://www.docker.com/products/docker-desktop/.

If you are working in WSL, you can follow the instructions below to set up your environment:

1. Install Docker Desktop. After installation, restart is needed, then open Docker Desktop, in Settings, Ressources, WSL Integration, Enable integration with additional distros – Ubuntu
2. Clone autogen and export `AUTOGEN_REPO_BASE`. This environment variable enables the Docker containers to use the correct version agents. `bash git clone git@github.com:microsoft/autogen.git export AUTOGEN_REPO_BASE=<path_to_autogen>`

Installation and Setup

[Deprecated currently] **To get the most out of AutoGenBench, the `agbench` package should be installed**. At present, the easiest way to do this is to install it via `pip`.

If you would prefer working from source code (e.g., for development, or to utilize an alternate branch), simply clone the AutoGen repository, then install `agbench` via:

```
pip install -e autogen/python/packages/agbench
```

After installation, you must configure your API keys. As with other AutoGen applications, AutoGenBench will look for the OpenAI keys in the OAI_CONFIG_LIST file in the current working directory, or the OAI_CONFIG_LIST environment variable. This behavior can be overridden using a command-line parameter described later.

If you will be running multiple benchmarks, it is often most convenient to leverage the environment variable option. You can load your keys into the environment variable by executing:

```
export OAI_CONFIG_LIST=$(cat ./OAI_CONFIG_LIST)
```

If an OAI_CONFIG_LIST is *not* provided (by means of file or environment variable), AutoGenBench will use the OPENAI_API_KEY environment variable instead.

For some benchmark scenarios, additional keys may be required (e.g., keys for the Bing Search API). These can be added to an `ENV.json` file in the current working folder. An example `ENV.json` file is provided below:

```
{
    "BING_API_KEY": "xxxyyyzzz"
}
```

A Typical Session

Once AutoGenBench and necessary keys are installed, a typical session will look as follows:

Navigate to HumanEval

```
cd autogen/python/packages/agbench/benchmarks/HumanEval
```

Note: The following instructions are specific to the HumanEval benchmark. For other benchmarks, please refer to the README in the respective benchmark folder, e.g.,: AssistantBench.

Create a file called ENV.json with the following (required) contents (If you're using MagenticOne), if using Azure:

```
{
    "CHAT_COMPLETION_KWARGS_JSON": "{}",
    "CHAT_COMPLETION_PROVIDER": "azure"
}
```

You can also use the openai client by replacing the last two entries in the ENV file by:

- `CHAT_COMPLETION_PROVIDER='openai'`
- `CHAT_COMPLETION_KWARGS_JSON` with the following JSON structure:

```
{
  "api_key": "REPLACE_WITH_YOUR_API",
  "model": "REPLACE_WITH_YOUR_MODEL"
}
```

Now initialize the tasks.

```
python Scripts/init_tasks.py
```

Note: This will attempt to download HumanEval

Once the script completes, you should now see a folder in your current directory called `Tasks` that contains one JSONL file per template in `Templates`.

Now to run a specific subset of HumanEval use:

```
agbench run Tasks/human_eval_MagenticOne.jsonl
```

You should see the command line print the raw logs that shows the agents in action To see a summary of the results (e.g., task completion rates), in a new terminal run the following:

```
agbench tabulate Results/human_eval_MagenticOne
```

Where:

- `agbench run Tasks/human_eval_MagenticOne.jsonl` runs the tasks defined in `Tasks/human_eval_MagenticOne.jsonl`
- `agbench tablue results/human_eval_MagenticOne` tabulates the results of the run

Each of these commands has extensive in-line help via:

- `agbench --help`
- `agbench run --help`
- `agbench tabulate --help`
- `agbench remove_missing --help`

NOTE: If you are running `agbench` from within the repository, you need to navigate to the appropriate scenario folder (e.g., `scenarios/HumanEval`) and run the `Scripts/init_tasks.py` file.

More details of each command are provided in the sections that follow.

Running AutoGenBench

To run a benchmark (which executes the tasks, but does not compute metrics), simply execute:

```
cd [BENCHMARK]
agbench run Tasks/*.jsonl
```

For example,

```
cd HumanEval
agbench run Tasks/human_eval_MagenticOne.jsonl
```

The default is to run each task once. To run each scenario 10 times, use:

```
agbench run --repeat 10 Tasks/human_eval_MagenticOne.jsonl
```

The agbench command-line tool allows a number of command-line arguments to control various parameters of execution. Type agbench -h to explore these options:

```
'agbench run' will run the specified autogen scenarios for a given number of repetitions and record all logs and trace
     information. When running in a Docker environment (default), each run will begin from a common, tightly controlled,
     environment. The resultant logs can then be further processed by other scripts to produce metrics.

positional arguments:
  scenario      The JSONL scenario file to run. If a directory is specified,
                then all JSONL scenarios in the directory are run. (default:
                ./scenarios)

options:
  -h, --help            show this help message and exit
  -c CONFIG, --config CONFIG
                        The environment variable name or path to the OAI_CONFIG_LIST (default: OAI_CONFIG_LIST).
  -r REPEAT, --repeat REPEAT
                        The number of repetitions to run for each scenario (default: 1).
  -s SUBSAMPLE, --subsample SUBSAMPLE
                        Run on a subsample of the tasks in the JSONL file(s). If a decimal value is specified, then run on
                        the given proportion of tasks in each file. For example "0.7" would run on 70% of tasks, and "1.0"
                        would run on 100% of tasks. If an integer value is specified, then randomly select *that* number of
                        tasks from each specified JSONL file. For example "7" would run tasks, while "1" would run only 1
                        task from each specified JSONL file. (default: 1.0; which is 100%)
  -m MODEL, --model MODEL
                        Filters the config_list to include only models matching the provided model name (default: None, which
                        is all models).
  --requirements REQUIREMENTS
                        The requirements file to pip install before running the scenario.
  -d DOCKER_IMAGE, --docker-image DOCKER_IMAGE
                        The Docker image to use when running scenarios. Can not be used together with --native. (default:
                        'agbench:default', which will be created if not present)
  --native              Run the scenarios natively rather than in docker. NOTE: This is not advisable, and should be done
                        with great caution.
```

Results

By default, the AutoGenBench stores results in a folder hierarchy with the following template:

`./results/[scenario]/[task_id]/[instance_id]`

For example, consider the following folders:

`./results/default_two_agents/two_agent_stocks/0` `./results/default_two_agents/two_agent_stocks/1`

...

`./results/default_two_agents/two_agent_stocks/9`

This folder holds the results for the `two_agent_stocks` task of the `default_two_agents` tasks file. The `0` folder contains the results of the first instance / run. The `1` folder contains the results of the second run, and so on. You can think of the *task_id* as mapping to a prompt, or a unique set of parameters, while the *instance_id* defines a specific attempt or run.

Within each folder, you will find the following files:

- *timestamp.txt*: records the date and time of the run, along with the version of the autogen-agentchat library installed
- *console_log.txt*: all console output produced by Docker when running AutoGen. Read this like you would a regular console.
- *[agent]_messages.json*: for each Agent, a log of their messages dictionaries
- *./coding*: A directory containing all code written by AutoGen, and all artifacts produced by that code.

Contributing or Defining New Tasks or Benchmarks

If you would like to develop – or even contribute – your own tasks or benchmarks, please review the contributor's guide for complete technical details.