Auto Test Configuration User Guide

Introduction

This manual provides instructions for using **Auto Test Configuration**, the module responsible for defining and managing automated test settings.

It explains how to:

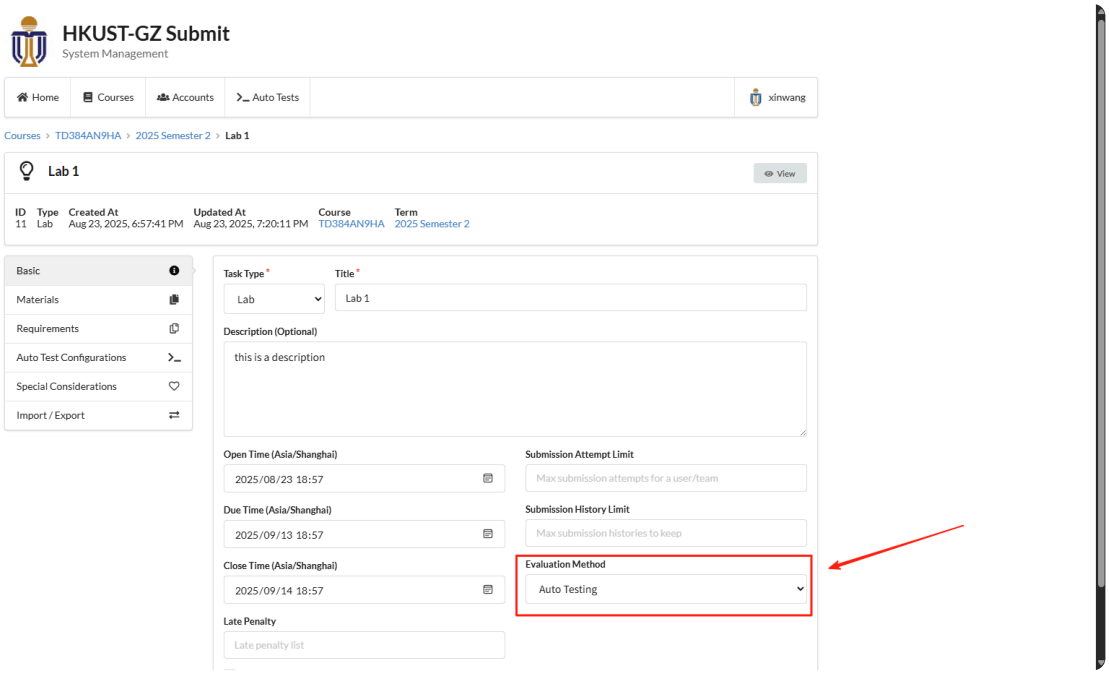
- Create and edit test configurations
- Define file requirements and test environments
- Set scoring and execution rules

Step 1: Enable Auto Testing in Basic Settings

Before creating an Auto Test Configuration, you must enable **Auto Testing** as the evaluation method in the task's **Basic** tab.

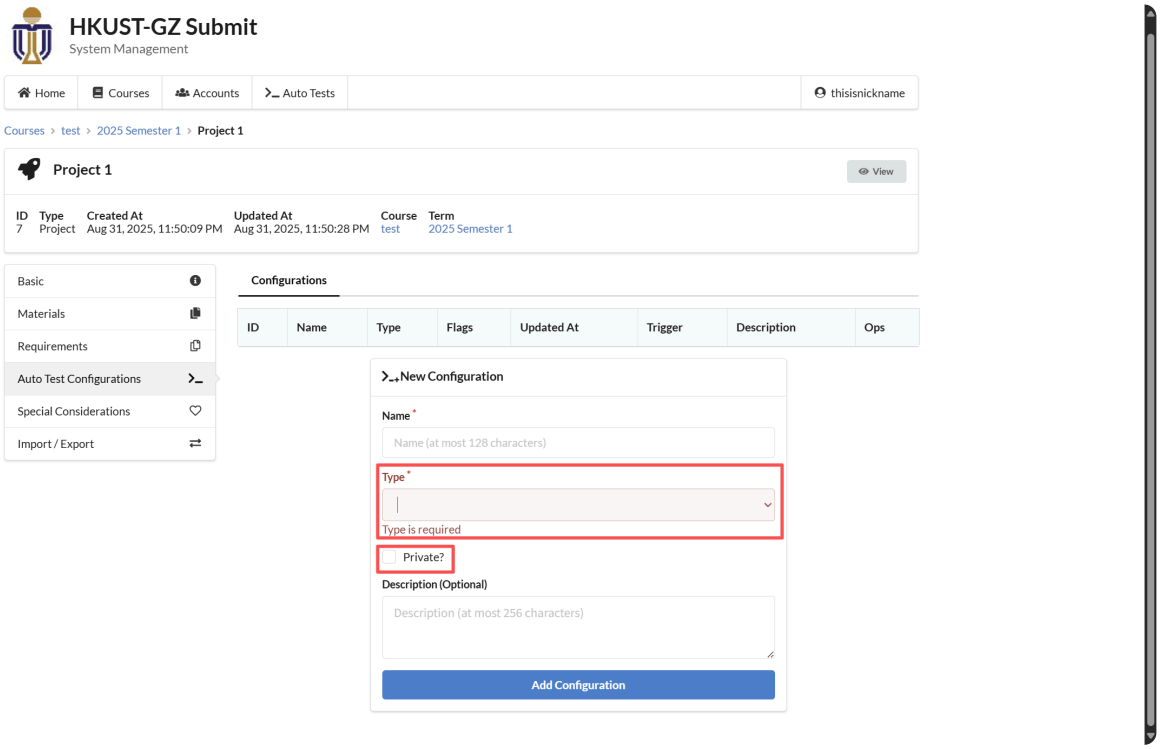
1. Open the task (e.g., Lab 1).
2. In the **Basic** settings, locate **Evaluation Method**.
3. Select **Auto Testing** from the dropdown menu.
4. Save the task.

Only after enabling Auto Testing here can you proceed to define detailed configurations in the **Auto Test Configurations** tab.



Step 2: Create a New Auto Test Configuration

1. Go to the **Auto Test Configurations** tab.
2. Enter a **Name** and select a **Type** (e.g., Anti Plagiarism, Docker, File Exists, Run Script and Online Judge).
3. Use **Private** carefully depending on whether students should see plagiarism reports.
4. Click **Add Configuration** to save.
5. After creation, click the **Edit** button (✏️) under **Ops** to open the configuration editor.

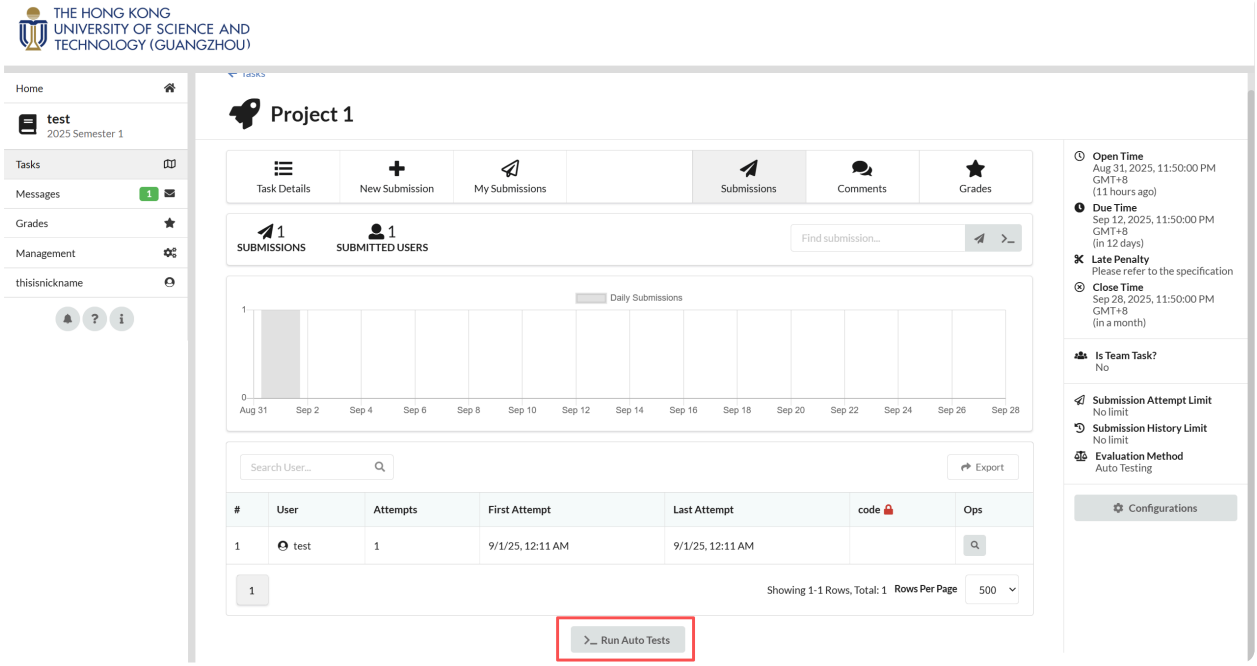


Step 3: Run Auto Tests

If the configuration is set to **Auto Trigger**, tests will run automatically whenever a student submits.

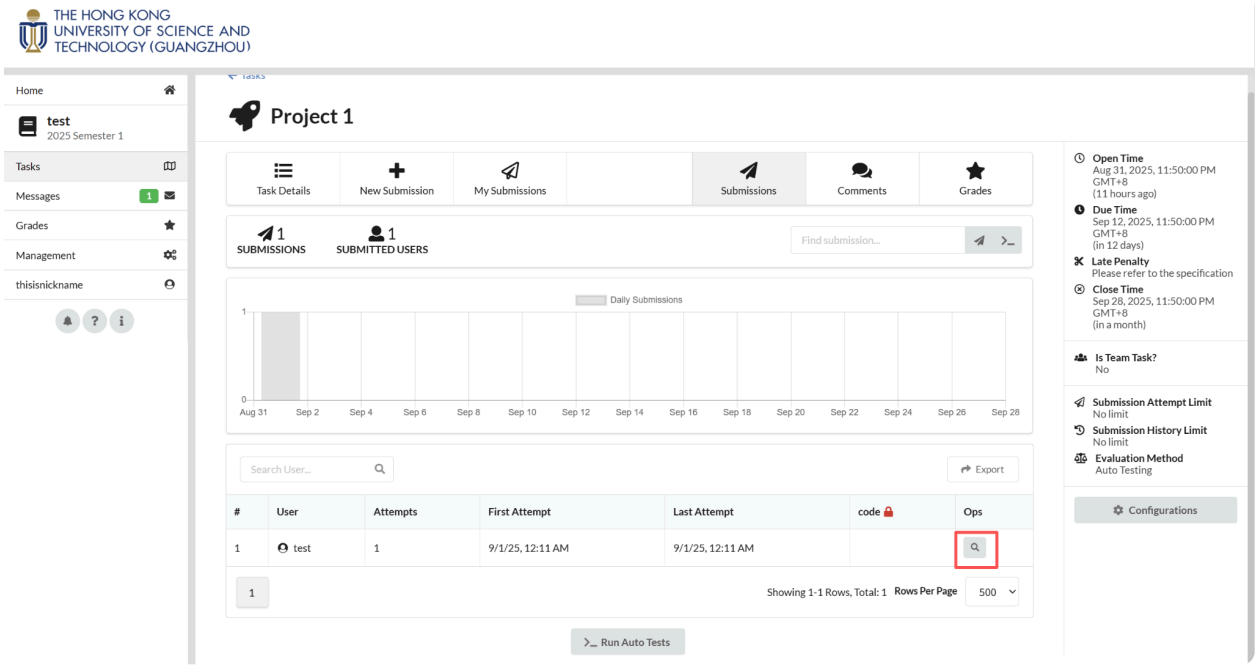
If the configuration is set to **Manual Trigger**, you can execute tests in two ways:

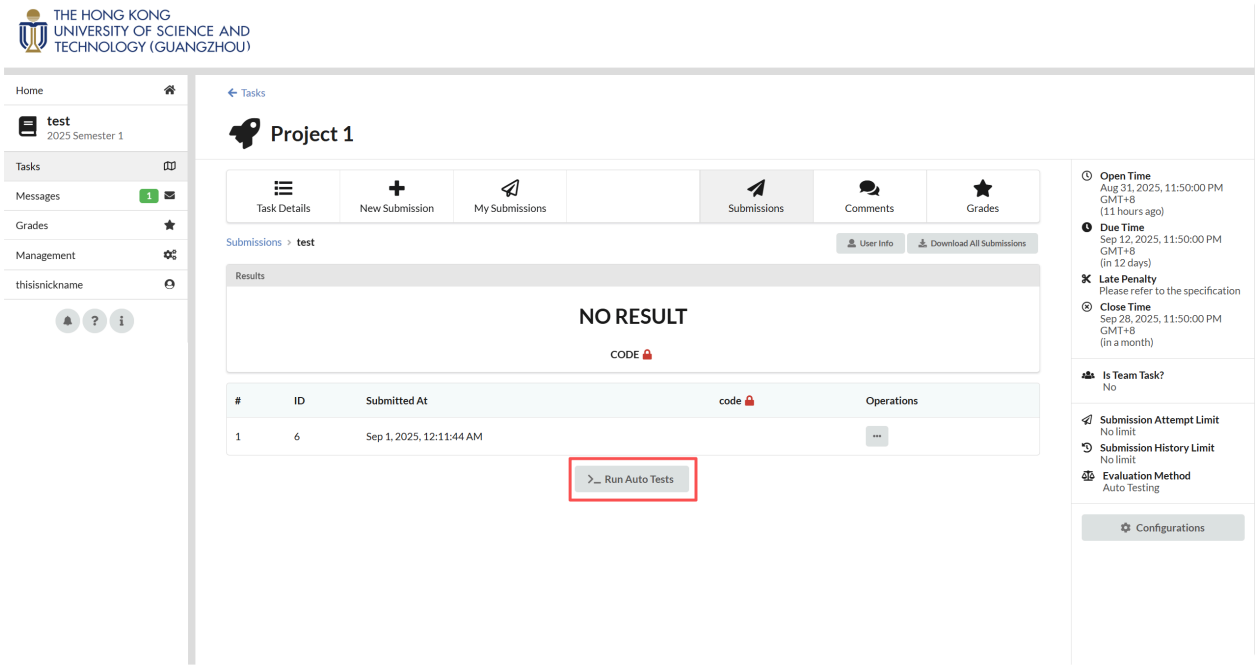
1. **Run for All Submissions**
   - Go to the task's **Submissions** page.
   - Click **Run Auto Tests** at the bottom (see red box below).
   - This will trigger the configured tests for **all submissions** under the task.



2. **Run for a Single Submission**
   - On the **Submissions** page, click the **Ops → View (magnifier icon)** for the target student.
   - This opens the submission detail page.

- Click **Run Auto Tests** to execute the configured tests only for this submission.





Configuration Types

Here are the available types of Auto Test Configurations.

Online Judge (OJ)

The Online Judge (OJ) automatically runs different types of tests based on the selected **Test Environment** and provides AI-generated feedback.

Before configuring OJ tests, you must first upload the required **Test Environment file** under **Materials**.
The specific settings of the test environment vary depending on the testing requirements.

Currently, two main categories of OJ tests are supported:

- **Code Testing**: Executes student code in a controlled environment and evaluates correctness, efficiency, and other metrics.
- **Q&A Scoring**: Grades answers to open-ended or essay-style questions, with AI feedback for improvement.

Code Testing

Code Testing runs student submissions against predefined input/output cases and provides both a score and AI-generated feedback.

Steps to Configure

1. **Prepare Test Cases**
    - Create a folder named `cases`.
    - Inside this folder, add multiple input/output pairs:
        - `input0.txt`, `output0.txt`
        - `input1.txt`, `output1.txt`
        - `input2.txt`, `output2.txt`
        - and so on.

2. **Package Test Environment**
    - Compress the entire `cases` folder into a `.zip` archive (e.g., `cases.zip`).
    - Go to **Materials** and upload the `.zip` file with type **Test Environment**.
    - It is strongly recommended to enable **Private**, otherwise the test cases will be visible to students.

3. **Add Auto Test Configuration**
    - In **Auto Test Configurations**, create a new configuration with type **Online Judge**.
    - Select the uploaded `cases.zip` as the test environment.
    - Save the configuration.

4. **Run and View Results**

- Once enabled, the OJ will automatically run submissions against the uploaded test cases.
- The frontend will display the **score** and **feedback** (including analysis, suggestions, and areas for improvement).
- If the configuration is not marked as **Private**, the results will be visible to students.
- ⚠️ **Important**:
  The OJ automatically detects and runs either `.cpp` or `.py` files in a submission. If multiple code files are included, it may cause conflicts. To avoid confusion, each task should be created as a separate **Lab / Assignment / Project**.

Notes

- Always ensure that input/output file names strictly match (`inputN.txt` ↔ `outputN.txt`).
- Keeping the environment private is recommended to prevent students from accessing the raw test cases.

Q&A Scoring

Q&A Scoring evaluates open-ended or essay-style answers and provides AI-generated feedback and scoring based on reference materials.

Steps to Configure

1. **Prepare QA Folder**
   - Create a folder named **`qa`** .
   - The folder must contain the following files:
     - **`name.txt`** : The question title.
     - **`description.txt`** : Additional description or background for the question.
     - **`answer.txt`** : The standard or reference answer.

- **`comment_req.txt`** : (Optional) Special requirements for feedback. Must be included even if empty.
- **`score_req.txt`** : (Optional) Special requirements for scoring. Must be included even if empty.

⚠️ **Note**: The first three files (`name.txt` , `description.txt` , `answer.txt`) are mandatory.
The last two are optional but must exist in the folder (can be empty).

2. **Package Test Environment**
   - Compress the `qa` folder into a `.zip` archive (e.g., `qa.zip`).
   - Go to **Materials** and upload the archive with type **Test Environment**.
   - It is recommended to set **Private**, otherwise the reference answers will be visible to students.

3. **Add Auto Test Configuration**
   - In **Auto Test Configurations**, create a new configuration with type **Online Judge**.
   - Select the uploaded `qa.zip` as the test environment.
   - Save the configuration.

4. **Run and View Results**
   - Once enabled, the OJ will automatically compare student answers with the reference materials.
   - The frontend will display the **score** and **feedback**, including analysis, suggestions, and improvement points.
   - If the configuration is not marked as **Private**, the results will be visible to students.
   - ⚠️ **Important**:
     For Q&A tasks, the OJ automatically looks for a **`.md` file** in the student's submission as the answer.
     Make sure students are instructed to submit their responses in Markdown format.

Notes

- Always ensure the required files are present; missing `comment_req.txt` or `score_req.txt` will cause errors, even if not used.

- Keeping the environment private is strongly recommended to prevent exposing the standard answers.

Anti Plagiarism

The **Anti Plagiarism** configuration checks submission similarity to detect potential code copying basing AST.

Key Settings

- **Target File Requirement**: Select the file that will be compared (e.g., `code.py`). This must be specified for the plagiarism check to run.
- **Private**: If enabled, the test results will not be visible to students. This is recommended when the instructor wants to hide similarity details.
- **Trigger**: Defines when the test will run.
  - *Manual*: The test must be started manually.
  - *Automatic*: The test is triggered automatically upon submission.

Notes

- Ensure the correct **Target File Requirement** is selected, otherwise the configuration will not execute.

Viewing Results

After the test runs, results are shown under the **Submissions** tab:

- A summary result (e.g., "*Weak Evidence*") will be displayed.
- Click the **Ops** button to view detailed information.
- Detailed output includes similarity metrics such as:
  - **Conclusion** (e.g., Weak Evidence / Strong Evidence)
  - **Coverage** (percentage of detected similarity)

- **Potentially similar submission IDs**

Run Script

The **Run Script** configuration allows fully customized testing by executing a user-provided shell script.

How It Works

- The system will execute a file named `run.sh`.
- Users must prepare this script along with all required files, and package them into a single **.zip** archive.
- The archive must be uploaded to **Materials** as test environment and linked in the configuration.

Output Format (Important)

The system parses results **only from standard output**, and **only lines containing `${RESULT_TAG}` will be returned to the frontend**. A JSON object is recommended.

If you need to return more than one value (e.g., score, time, status), it is strongly recommended to output them as a single JSON object with one ${RESULT_TAG} prefix, instead of printing multiple tagged lines.

Your script must print the result in the following format:

```
echo "${RESULT_TAG} {\"score\": ${score}}"
```

Viewing Results

After the test runs, results are shown under the **Submissions** tab:

By specifying a **Result Conclusion Path** in the Auto Test Configuration, the system will extract only the selected field (e.g., `score`) from the JSON output, making the frontend display more concise.

Docker

The **Docker** configuration allows automated testing to run inside a fully controlled container environment.
By packaging a `Dockerfile` and related files, instructors can define a consistent runtime environment for all submissions, ensuring reproducibility and isolation.

This method is suitable when:

- Specific dependencies or libraries must be preinstalled.
- The testing process requires a customized runtime setup.
- Code needs to be executed in an environment closer to production.

Compared to other configuration types, Docker provides maximum flexibility, as you have full control over the build and execution process inside the container.

What You Need to Prepare

Before using the Docker configuration, you must create a **Test Environment** archive and upload it to **Materials**.

1. **File Structure**
   Prepare a folder with the following layout:

```
docker.zip/
├─ Dockerfile # required, defines the build environment
├─ .dockerignore # optional, recommended to reduce image size
└─ test/
    ├─ requirements.txt # dependencies to install
    ├─ main.py # the main entry script
    └─ ... # any other resources
```

2. **Archive**

- Compress the folder into a `.zip` file (e.g., `docker.zip`).
- The archive must contain the `Dockerfile` at the top level.

3. **Upload**

- Go to **Materials → New Material**.
- Select **Material Type = Test Environment**.
- Upload the prepared `docker.zip`.
- It is recommended to set the material as **Private**, otherwise the test environment will be visible to students.

⚠️ **Note**:

> The `main.py` script must print results with the required tag.
> Only lines containing `${RESULT_TAG}` will be parsed and returned to the frontend.
> Json object is recommended.

Example Dockerfile

```
# Base image (use a stable LTS version)
FROM ubuntu:18.04

# Set default locale
ENV LANG C.UTF-8

# Install system utilities
RUN apt-get update && apt-get install -y --no-install-recommends \
    ca-certificates wget \
 && rm -rf /var/lib/apt/lists/*

# Install Miniconda
RUN wget --quiet -O /usr/local/miniconda.sh \
    https://repo.continuum.io/miniconda/Miniconda3-4.5.12-Linux-x86_64.sh \
 && bash /usr/local/miniconda.sh -b -p /usr/local/miniconda \
 && rm /usr/local/miniconda.sh
ENV PATH /usr/local/miniconda/bin:$PATH

# Create Python environment
RUN conda create -y -n test python=3.6.5
ENV PATH /usr/local/miniconda/envs/test/bin:$PATH

# Set working directory
WORKDIR /root/test

# Copy test environment
COPY ./test /root/test

# Install Python dependencies
# (you may replace with a mirror source for faster download in China, e.g. Tsin
RUN pip install --no-cache-dir -r requirements.txt

# Entry point with timeout control (limits execution time)
ENTRYPOINT ["timeout", "-k", "10s", "1m"]

# Default command: run evaluation script
CMD ["python", "main.py"]
```

Result Processing

In the Docker configuration, you can customize how the results are shown on the frontend.

- **Result Render HTML**

  Allows you to provide an HTML template to render test results.

  If left empty, the raw JSON output will be displayed.

  By writing a simple HTML structure, you can highlight scores, status, or messages in a more user-friendly format.

- **Result Conclusion Path**
  Defines the JSON field to extract as the main conclusion (e.g., `score`).
  If empty, the entire JSON object will be used as the conclusion.
  This helps keep the frontend display concise by showing only the key field.

> **Tip**: Combining both options makes the frontend output much clearer and more readable.