EvalAI Documentation

Release 1.1

CloudCV Team

Jul 10, 2025

CONTENTS

EvalAI is an open source platform for evaluating and comparing machine learning (ML) and artificial intelligence algorithms (AI) at scale.

It is built to provide a scalable solution to the research community to fulfill the critical need of evaluating machine learning models and agents acting in an environment against annotations or with a human-in-the-loop.

Contents:

CHAPTER

ONE

GETTING STARTED

1.1 Introduction

EvalAI aims to build a centralized platform to host, participate, and collaborate in Artificial Intelligence (AI) challenges organized around the globe and hope to help in benchmarking progress in AI.

1.1.1 Features

Custom evaluation protocol

We allow creation of an arbitrary number of evaluation phases and dataset splits, compatibility using any programming language, and organizing results in both public and private leaderboards.

Remote evaluation

Certain large-scale challenges need special compute capabilities for evaluation. If the challenge needs extra computational power, challenge organizers can easily add their own cluster of worker nodes to process participant submissions while we take care of hosting the challenge, handling user submissions, and maintaining the leaderboard.

Evaluation inside RL environments

EvalAI lets participants submit code for their agent in the form of docker images which are evaluated against test environments on the evaluation server. During evaluation, the worker fetches the image, test environment, and the model snapshot and spins up a new container to perform evaluation.

CLI support

EvalAI-CLI is designed to extend the functionality of the EvalAI web application to your command line to make the platform more accessible and terminal-friendly.

Portability

EvalAI was designed with keeping in mind scalability and portability of such a system from the very inception of the idea. Most of the components rely heavily on open-source technologies – Docker, Django, Node.js, and PostgreSQL.

Faster evaluation

We warm-up the worker nodes at start-up by importing the challenge code and pre-loading the dataset in memory. We also split the dataset into small chunks that are simultaneously evaluated on multiple cores. These simple tricks result in faster evaluation and reduces the evaluation time by an order of magnitude in some cases.

1.2 Installation

1.2.1 Dependencies

EvalAI can run on Linux, Cloud, Windows, and macOS platforms. Please install docker and docker-compose before getting started with the installation of EvalAI.

1.2.2 Installation instructions

Once you have installed docker and docker-compose, please follow these steps to setup EvalAI on your local machine.

1. Get the source code on to your machine via git

```
git clone https://github.com/Cloud-CV/EvalAI.git evalai && cd "$_"
```

2. Build and run the Docker containers. This might take a while.

```
docker-compose up --build
```

3. That's it. Open web browser and hit the URL http://127.0.0.1:8888. Three users will be created by default which are listed below:

If you are facing any issue during installation, please see our common errors during installation page.

1.3 Setup

1.3.1 Docker Setup

Guide to setup EvalAI using Docker.

1.3.2 Manual Setup

Guide to setup EvalAI manually.

1.3.3 Linux Setup

Guide to setup EvalAI on Linux.

1.3.4 Windows Setup

Guide to setup EvalAI on Windows.

1.3.5 MacOS Setup

Guide to setup EvalAI on MacOS.

1.4 Pricing

Thank you for your interest in hosting your challenge on EvalAI. We're excited to support your work. We offer 4 plans based on the compute needs of the challenge hosts. Each plan is a monthly subscription (managed via Stripe) with no long-term commitment—you can cancel anytime once your challenge concludes. Stripe accepts all major payment methods and provides monthly receipts.

Each plan is a monthly subscription that includes one always-on worker to handle submissions (except for the Remote Evaluation Plan, which relies on your own infrastructure). If you expect higher submission volume—especially near

deadlines—we can scale easily by adding additional workers. These extra workers will be billed separately according to your selected plan.

Below is a detailed comparison of the available plans:

If you're unsure about which plan is the best fit for your challenge, we recommend starting with the Essentials plan and upgrading as needed.

Please feel free to reach out at team@eval.ai if you have any questions or would like to discuss further.

CHAPTER

TWO

FOR CHALLENGE HOSTS

2.1 Hosting Guide

2.1.1 Getting Started

2.1.2 Challenge Types

Prediction upload based challenges: Participants upload predictions corresponding to ground truth labels in the form of a file (could be any format: `json`, `npy`, `csv`, `txt` etc.)

Some of the popular prediction upload based challenges that we have hosted are shown below:

If you are interested in hosting prediction upload based challenges, then click here.

2.1.3 Host Challenge

EvalAI supports hosting challenges with different configurations. Challenge organizers can choose to customize most aspects of the challenge but not limited to:

- Evaluation metrics
- Language/Framework to implement the metric
- Number of phases and data-splits
- Daily / monthly / overall submission limit
- Number of workers evaluating submissions
- Evaluation on remote machines
- Show / hide error bars on leaderboard
- Public / private leaderboards
- Allow / block certain email addresses to participate in the challenge or phase
- Choose which fields to export while downloading challenge submissions

We have hosted challenges from different domains such as:

- Machine learning (2019 SIOP Machine Learning Competition)
- Deep learning (Visual Dialog Challenge 2019)
- Computer vision (Vision and Language Navigation)
- Natural language processing (VQA Challenge 2019)
- Healthcare (fastMRI Image Reconstruction)

- Self-driving cars (CARLA Autonomous Driving Challenge)

Host challenge using github

Step 1: Use template

Use EvalAI-Starters template. See this on how to use a repository as template.

Step 2: Generate github token

Generate your github personal access token and copy it in clipboard.

Add the github personal access token in the forked repository's secrets with the name `AUTH_TOKEN`.

Step 3: Setup host configuration

Now, go to EvalAI to fetch the following details -

1. `evalai_user_auth_token` - Go to profile page after logging in and click on `Get your Auth Token` to copy your auth token.
2. `host_team_pk` - Go to host team page and copy the `ID` for the team you want to use for challenge creation.
3. `evalai_host_url` - Use `https://eval.ai` for production server and `https://staging.eval.ai` for staging server.

Step 4: Setup automated update push

Create a branch with name `challenge` in the forked repository from the `master` branch. Note: Only changes in `challenge` branch will be synchronized with challenge on EvalAI.

Add `evalai_user_auth_token` and `host_team_pk` in `github/host_config.json`.

Step 5: Update challenge details

Read EvalAI challenge creation documentation to know more about how you want to structure your challenge. Once you are ready, start making changes in the yaml file, HTML templates, evaluation script according to your need.

Step 6: Push changes to the challenge

Commit the changes and push the `challenge` branch in the repository and wait for the build to complete. View the logs of your build.

If challenge config contains errors then a `issue` will be opened automatically in the repository with the errors otherwise the challenge will be created on EvalAI.

Step 7: Verify challenge

Go to Hosted Challenges to view your challenge. The challenge will be publicly available once EvalAI admin approves the challenge.

To update the challenge on EvalAI, make changes in the repository and push on `challenge` branch and wait for the build to complete.

Host prediction upload based challenge

Step 1: Setup challenge configuration

We have created a sample challenge configuration that we recommend you to use to get started. Use EvalAI-Starters template to start. See this on how to use a repository as template.

Step 2: Edit challenge configuration

Open `challenge_config.yml` from the repository that you cloned in step-1. This file defines all the different settings of your challenge such as start date, end date, number of phases, and submission limits etc.

Edit this file based on your requirement. For reference to the fields, refer to the challenge configuration reference section.

Step 3: Edit evaluation script

Next step is to edit the challenge evaluation script that decides what metrics the submissions are going to be evaluated on for different phases.

Please refer to the writing evaluation script to complete this step.

Step 4: Edit challenge HTML templates

Almost there. You just need to update the HTML templates in the `templates/` directory of the bundle that you cloned.

EvalAI supports all kinds of HTML tags which means you can add images, videos, tables etc. Moreover, you can add inline CSS to add custom styling to your challenge details.

Congratulations! you have submitted your challenge configuration for review and EvalAI team has notified about this. EvalAI team will review and will approve the challenge.

If you have issues in creating a challenge on EvalAI, please feel free to contact us at team@cloudcv.org create an issue on our GitHub issues page.

Host a remote evaluation challenge

Step 1: Set up the challenge

Follow host challenge using github section to set up a challenge on EvalAI.

Step 2: Edit challenge configuration

Set the `remote_evaluation` parameter to `True` in `challenge_config.yaml`. This challenge config file defines all the different settings of your challenge such as start date, end date, number of phases, and submission limits etc.

Edit this file based on your requirement. For reference to the fields, refer to the challenge configuration reference section.

Please ensure the following fields are set to the following values:

- `remote_evaluation : True`

Refer to the following documentation for details on challenge configuration.

Step 3: Edit remote evaluation script

Next step is to edit the challenge evaluation script that decides what metrics the submissions are going to be evaluated on for different phases. Please refer to Writing Remote Evaluation Script section to complete this step.

Step 4: Set up remote evaluation worker

1. Create conda environment to run the evaluation worker. Refer to conda's create environment section to set up a virtual environment.
2. Install the worker requirements from the `EvalAI-Starters/remote_challenge_evaluation` present here:

```
cd EvalAI-Starters/
pip install remote_challenge_evaluation/requirements.txt
```

3. Start evaluation worker:

```
cd EvalAI-Starters/remote_challenge_evaluation
python main.py
```

If you have issues in creating a challenge on EvalAI, please feel free to contact us at team@cloudcv.org create an issue on our GitHub issues page.

2.1.4 Approval Process

Note: If you are hosting the challenge on eval.ai, then you cannot approve your challenge. It will be approved by EvalAI team. You can skip this section.

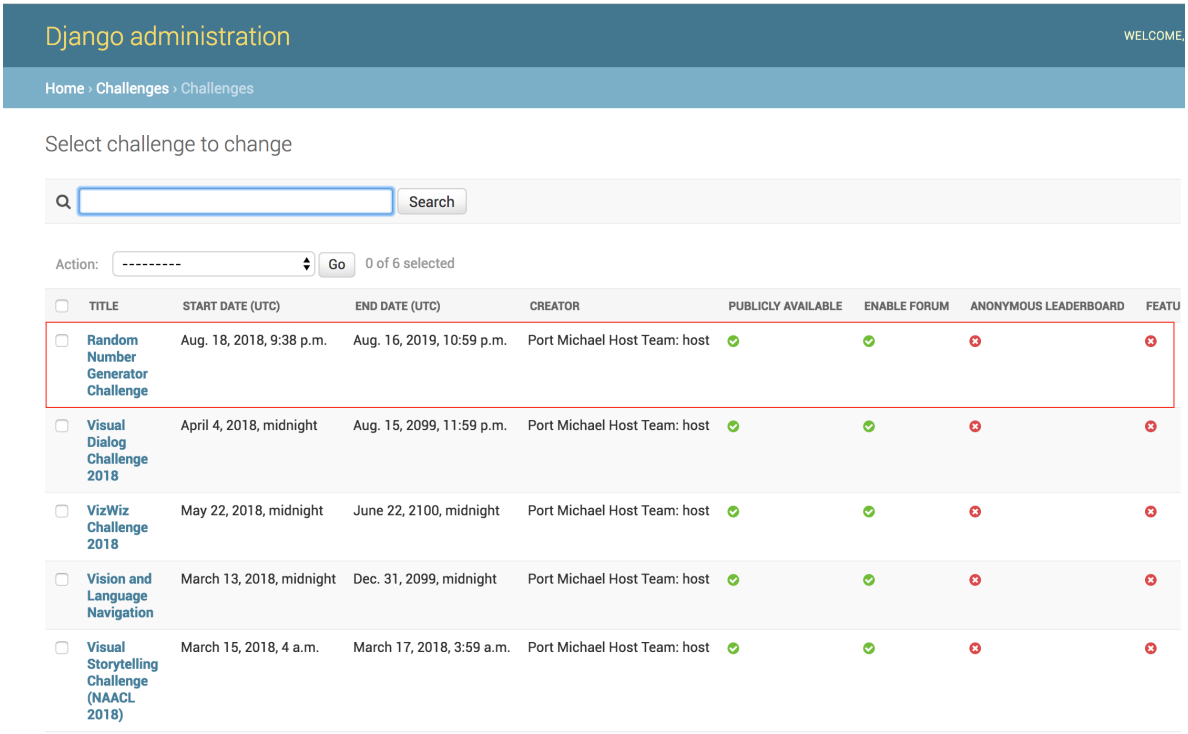
Approve a challenge (for forked version)

Once a challenge config has been uploaded, the challenge has to be approved by the EvalAI Admin (i.e. you if you are setting up EvalAI yourself on your server) to make it available to everyone. Please follow the following steps to approve a challenge (if you are):

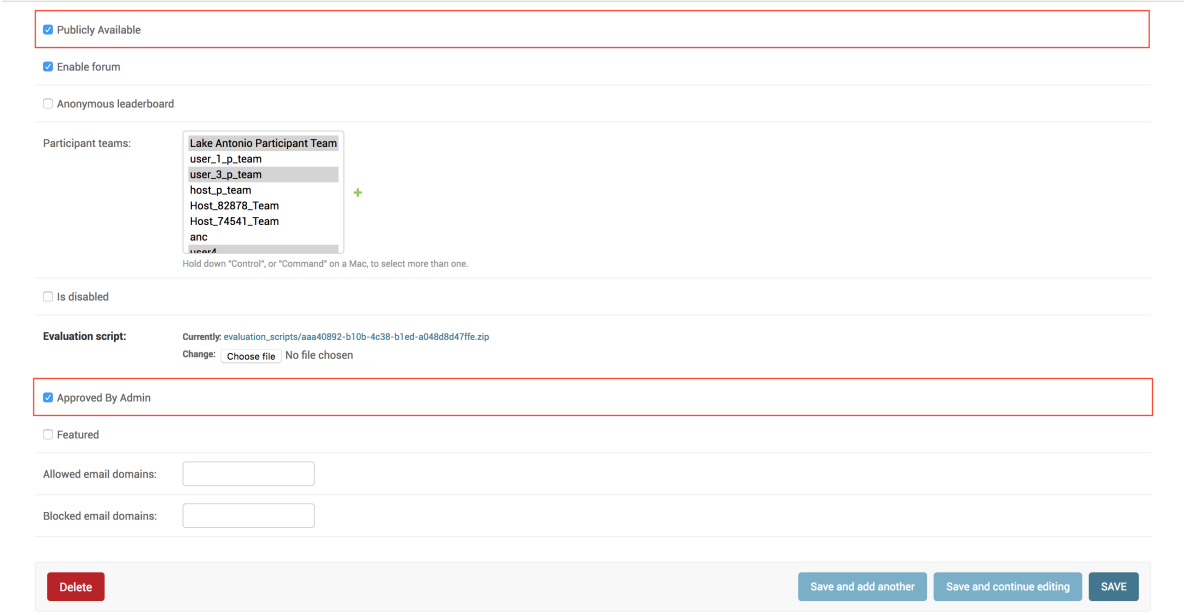
Let's assume that we want to approve a challenge with name `Random Number Generator Challenge`.

Step 1: Approve challenge using django admin

1. Login to EvalAI's django admin panel, and you will see the list of challenges



2. Click on the challenge that you want to approve and scroll to bottom to check the following two fields.
    - Approved By Admin
    - Publically Available



Now, save the challenge. The challenge has been successfully approved by the administrator and is also publicly visible to the users.

Step 2: Reload submission worker

Since you have just approved the challenge, the submission worker has to be reloaded so that it can fetch the evaluation script and other related files for your challenge from the database. Now reload the submission worker using the following command:

Run the following command:

```
docker-compose restart worker
```

Submission worker has been successfully reloaded!

Now, the challenge is ready to accept submissions from participants.

If you have issues in hosting a challenge on forked version of EvalAI, please feel free to create an issue on our GitHub Issues Page.

2.2 Configuration

This guide is for challenge organizers who want to set up and customize challenges on EvalAI using YAML configuration files. Each section below explains a part of the config with examples and common use cases.

2.2.1 Challenge Configuration

This section explains how to configure the main details of your EvalAI challenge in the `challenge_config.yaml` file. It includes challenge metadata, display settings, timeline, tags, and top-level files (evaluation scripts, images, HTML templates).

For ready to use end-to-end challenge configuration examples refer to this section.

Following fields are required (and can be customized) in the `challenge_config.yml`.

Challenge Metadata

- **`title` (required)**

  Type: `string`

  Description: The full name of the challenge displayed to users.

  Example:

  ```
  title: "Autonomous Driving Lane Detection Challenge"
  ```

- **`short_description` (required)**

  Type: `string`

  Description: A short summary (~140 characters) of the challenge.

  Example:

  ```
  short_description: "Detect lane boundaries from images in real-time."
  ```

- **`description` (required)**

  Type: `string` (relative file path)

  Description: Path to the full challenge description in HTML file format.

Example:

```
description: "templates/description.html"
```

- **evaluation_details (required)**

  Type: `string` (relative file path)

  Description: Path to a detailed explanation of the evaluation process in HTML file format.

  Example:

```
evaluation_details: "templates/evaluation_details.html"
```

- **terms_and_conditions (required)**

  Type: `string` (relative file path)

  Description: Path to HTML file with challenge rules, licenses, restrictions, etc.

  Example:

```
terms_and_conditions: "templates/terms_and_conditions.html"
```

- **image (required)**

  Type: `string` (relative file path)

  Description: Path to the challenge logo. Must be `.jpg`, `.jpeg`, or `.png`.

  Example:

```
image: "images/logo/lane_detection_logo.png"
```

- **submission_guidelines (required)**

  Type: `string` (relative file path)

  Description: Path to HTML file with "how-to-submit" instructions.

  Example:

```
submission_guidelines: "templates/submission_guidelines.html"
```

Challenge Timeline

- **start_date (required)**

  Type: `datetime (UTC)`

  Format: `YYYY-MM-DD HH:MM:SS`

  Description: When the challenge opens.

  Example:

```
start_date: "2025-09-01 00:00:00"
```

- **`end_date` (required)**

  Type: `datetime (UTC)`

  Format: `YYYY-MM-DD HH:MM:SS`

  Description: When the challenge closes.

  Example:

```yaml
end_date: "2025-12-01 23:59:59"
```

Challenge Settings

- **`published` (optional)**

  Type: `boolean`

  Default: `False`

  Description: Whether the challenge should become publicly visible after EvalAI admin approval.

  Value:

  - `True`: Visible to all participants.
  - `False`: Hidden until you're ready to go live.

  Example:

```yaml
published: False
```

- **`remote_evaluation` (optional)**

  Type: `boolean`

  Default: `False`

  Description: Whether submissions will be evaluated on a remote machine.

  Value:

  - `True`: Evaluation will happen on external infrastructure you control.
  - `False`: EvalAI will handle evaluation in one of the paid plan tiers.

  Example:

```yaml
remote_evaluation: False
```

Tags

- **`tags` (optional)**

  Type: `list of strings`

  Description: Keywords used for displaying relevant areas of the challenge on the platform.

  Example:

```yaml
tags:
  - autonomous-driving
  - lane-detection
```

(continues on next page)

(continued from previous page)

```
  - computer-vision
  - real-time-processing
```

Evaluation Script

- **`evaluation_script` (required)**

  Type: `string` (relative file path)

  Description: Folder containing the python scripts that will be used to evaluate submissions.

  Example:

```
evaluation_script: "evaluation_script/"
```

To read more about evaluation scripts click here.

Leaderboard Configuration

- **`leaderboard_description` (optional)**

  Type: `string`

  Description: This is the description that appears above the leaderboard table on the challenge's leaderboard page. It can explain what the leaderboard metrics mean, how the ranking works, or provide any other context you want participants to know when viewing the leaderboard.

  Example:

```
leaderboard_description: "The leaderboard shows the evaluation results of your␣
↪submissions based on accuracy and F1 score. The higher the score, the better your␣
↪model performs."
```

- **`leaderboard` (required)**

  Type: `list of objects` **Description**: Defines leaderboard structure and metrics used for ranking.

  A leaderboard for a challenge on EvalAI consists of following subfields:

  - **id**: Unique positive integer field for each leaderboard entry
  - **schema**: Schema field contains the information about the rows of the leaderboard. A schema contains two keys in the leaderboard:
    1. `labels`: Labels are the header rows in the leaderboard according to which the challenge ranking is done.
    2. `default_order_by`: This key decides the default sorting of the leaderboard based on one of the labels defined above.
    3. `metadata`: This field defines additional information about the metrics that are used to evaluate the challenge submissions.

  Example:

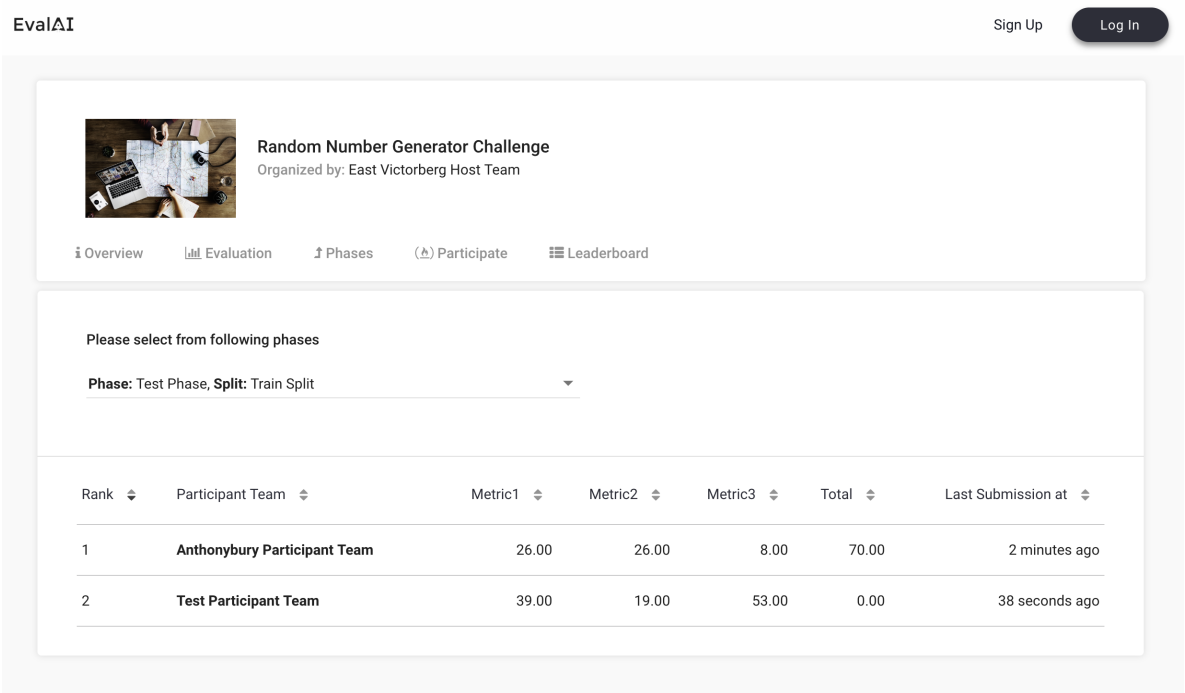
```
leaderboard:
- id: 1
  schema:
    {
```

(continues on next page)

(continued from previous page)

```
    "labels": ["Accuracy", "F1 Score", "Total"],
    "default_order_by": "Total",
    "metadata": {
      "Accuracy": {
        "sort_ascending": false,
        "description": "Overall accuracy of the model"
      },
      "F1 Score": {
        "sort_ascending": false,
        "description": "Weighted F1 score over all classes"
      },
      "Total": {
        "sort_ascending": false,
        "description": "Combined performance metric"
      }
    }
  }
```

The leaderboard schema will look something like this on leaderboard UI:



Example:

This is how the challenge configuration (excluding phases and splits configuration) of a sample challenge with all the above fields look like:

```
title: "Autonomous Driving Lane Detection Challenge"
short_description: "Detect lane boundaries from images in real-time."
description: "templates/description.html"
evaluation_details: "templates/evaluation_details.html"
terms_and_conditions: "templates/terms_and_conditions.html"
```

(continues on next page)

(continued from previous page)

```yaml
image: "images/logo/lane_detection_logo.png"
submission_guidelines: "templates/submission_guidelines.html"
leaderboard_description: "The leaderboard shows the evaluation results of your␣
↪submissions based on accuracy and F1 score. The higher the score, the better your␣
↪model performs."
evaluation_script: "evaluation_script/"
remote_evaluation: false
start_date: "2025-09-01 00:00:00"
end_date: "2025-12-01 23:59:59"
published: false
tags:
  - autonomous-driving
  - lane-detection
  - computer-vision
  - real-time-processing
leaderboard:
  - id: 1
    schema: {
      "labels": ["Accuracy", "F1 Score", "Total"],
      "default_order_by": "Total",
      "metadata": {
        "Accuracy": {
          "sort_ascending": false,
          "description": "Overall accuracy of the model"
        },
        "F1 Score": {
          "sort_ascending": false,
          "description": "Weighted F1 score over all classes"
        },
        "Total": {
          "sort_ascending": false,
          "description": "Combined performance metric"
        }
      }
    }
```

2.2.2 Challenge Phases Setup

Challenges on EvalAI can consist of multiple challenge phases, each representing a distinct stage in the evaluation lifecycle, such as development, validation, or testing. Each phase defines its own timeline, visibility settings, evaluation criteria, and submission rules.

Challenge phases are defined under the `challenge_phases` key in `challenge_config.yaml`.

Challenge Phase

Each challenge phase in a challenge contains the following subfields:

Phase Metadata

- **id (required)**

  Type: integer

  Description: Unique identifier for the phase. Must be positive and unique across all phases.

  Example:

```
id: 1
```

- **name (required)**

  Type: string

  Description: Display name for the phase, that will be shown to participants.

  Example:

```
name: "Dev Phase"
```

- **description (required) Type**: string (relative file path)

  Description: Path to a detailed description file for the phase in HTML format.

  Example:

```
description: "templates/challenge_phase_1_description.html"
```

- **challenge (required)**

  Type: integer

  Description: This field links the challenge phase to a specific challenge by its id. It's essential when defining multiple phases under different challenges in a single config file.

  Example:

```
challenge: 1
```

- **codename (required)**

  Type: string

  Description: Unique id for each challenge phase. Note that the codename of a challenge phase is used to map the results returned by the evaluation script to a particular challenge phase.

  Note: The codename specified here should match with the codename specified in the evaluation script to perfect mapping.

  Example:

```
codename: dev
```

How codename maps in the Evaluation Script?

When the submission is being evaluated, the evaluate() function in the evaluation script receives the codename like this:

```
def evaluate(test_annotation_file, user_annotation_file, phase_codename, **kwargs):
    if phase_codename == "dev":
        # Evaluation logic for Development Phase
        ...
    elif phase_codename == "test":
        # Logic for another phase (e.g., final test)
        ...
```

This mapping ensures that:

- The logic can vary across phases (e.g., different metrics, files).
- EvalAI can map results returned by evaluation script to the respective challenge phase accordingly.

To know more about how `codename` is used during Evaluation, read the guide on *Writing an Evaluation Script*.

Timeline

- **`start_date` (required)**

  Type: `datetime (UTC)`

  Format: `YYYY-MM-DD HH:MM:SS`

  Description: When the challenge phase opens.

  Example:

```
start_date: "2025-07-01 00:00:00"
```

- **`end_date` (required)**

  Type: `datetime (UTC)`

  Format: `YYYY-MM-DD HH:MM:SS`

  Description: When the challenge phase closes.

  Example:

```
end_date: "2025-09-01 23:59:59"
```

Phase Settings

- **`leaderboard_public` (optional)**

  Type: `boolean`

  Default: `False`

  Description: Whether to keep leaderboard public or private for this phase.

  Value:

  - `True`: Leaderboard is public and visible to all participants.
  - `False`: Leaderboard is private and not visible.

  Example:

```
leaderboard_public: True
```

- **`is_public` (optional)**

  Type: `boolean`

  Default: `False`

  Description: Defines whether the phase is visible to participants.

  Value:

  - `True`: Phase is visible to all participants.
  - `False`: Phase is private and not visible to participants.

  Example:

```
is_public: False
```

- **`is_active` (required)**

  Type: `boolean`

  Description: Specifies whether this challenge phase is currently active. EvalAI uses this field to determine which phase(s) is/are available to accept submissions.

  Value:

  - `True`: This phase is currently active.
  - `False`: This phase is inactive.

  Example:

```
is_active: True
```

- **`is_submission_public` (required)**

  Type: `boolean`

  Default: `False`

  Description: Defines whether the submissions are by default public or private. In case of public option, the participants won't be able to make it private.

  Note: This will only work when the `leaderboard_public` property is set to `True`.

  Value:

  - `True`: Submissions of this phase are public by default.
  - `False`: Submission of this phase are private by default.

  Example:

```
is_submission_public: True
```

- **`allowed_email_ids` (required)**

  Type: `list of strings`

  Description: A list of email IDs allowed to participate in the challenge. Leave blank if everyone is allowed to participate. (e.g. `["example1@domain1.com", "example2@domain2.org", "example3@domain3.com"]` Only the participants with these email ids will be allowed to participate.)

  Example:

```
allowed_email_ids: []
```

Submission Rules

- **max_submissions_per_day (optional)**

  Type: integer

  Default: 100000

  Description: Defines the maximum number of submissions allowed per day to a challenge phase.

  Example:

```
max_submissions_per_day: 100
```

- **max_submissions_per_month (optional)**

  Type: integer

  Default: 100000

  Description: Defines the maximum number of submissions allowed per month to a challenge phase.

  Example:

```
max_submissions_per_month: 2000
```

- **max_submissions (optional)**

  Type: integer

  Default: 100000

  Description: Defines the maximum number of total submissions that can be made to the challenge phase.

  Example:

```
max_submissions: 5000
```

- **max_concurrent_submissions_allowed (optional)**

  Type: integer

  Description: Max number of submissions allowed in parallel at any given time.

  Example:

```
max_concurrent_submissions_allowed: 5
```

- **allowed_submission_file_types (optional)**

  Type: string (String of comma separated ext.)

  Description: Restrict submission formats by file extension.

  Example:

```
allowed_submission_file_types: ".json, .zip, .csv, .tsv, .h5, .npy"
```

- **is_restricted_to_select_one_submission (optional)**

  Type: boolean

Default: `False`

Description: Defines whether to restrict a user to select only one submission for the leaderboard.

Example:

```
is_restricted_to_select_one_submission: True
```

Submission Metadata

- **default_submission_meta_attributes (optional)**

  Type: `list[Object]`

  Description: The default attributes that are shown for every submission. You can toggle their visibility.

  Example:

```
default_submission_meta_attributes:
  - name: method_name
    is_visible: True
  - name: method_description
    is_visible: True
  - name: project_url
    is_visible: True
  - name: publication_url
    is_visible: True
```

- **submission_meta_attributes (optional)**

  Type: `list[Object]`

  Description: These are custom metadata fields participant can add to their submission.

  Example:

```
submission_meta_attributes:
  - name: TextAttribute
    description: "Short text info"
    type: text
    required: False
  - name: SingleOptionAttribute
    description: "Choose one"
    type: radio
    options: ["A", "B", "C"]
  - name: MultipleChoiceAttribute
    description: "Select many"
    type: checkbox
    options: ["alpha", "beta", "gamma"]
  - name: TrueFalseField
    description: "Is this final?"
    type: boolean
    required: True
```

Evaluation Setup

- **test_annotation_file (required)**

  Type: `string` (relative file path)

  Description: The file that will be used for ranking the submission made by a participant. An annotation file can be shared by more than one challenge phase.

  Example:

```
test_annotation_file: "annotations/test_annotations_devsplit.json"
```

- **is_partial_submission_evaluation_enabled (optional)**

  Type: `boolean`

  Description: Defines whether the challenge workers should update Leaderboard Data for a specific Submission incrementally or all at once.

  Value:

  - `True`: Worker updates the leaderboard data incrementally to show the evaluation metric(s) as soon as they get computed.
  - `False`: Worker waits for all the metrics to get computed and show all of them at once.

  Example:

```
is_partial_submission_evaluation_enabled: False
```

Challenge Phase Splits

A challenge phase split defines the relationship between:

- A specific challenge phase (challenge_phase_id)
- A dataset split (dataset_split_id)
- A leaderboard (leaderboard_id)

This mapping allows the challenge hosts to control visibility, sorting, and presentation of submission results for each phase and split.

Mapping

- **challenge_phase_id (required)**

  Type: `integer`

  Description: ID of the challenge phase to map with (must match an entry in challenge_phases).

  Example:

```
challenge_phase_id: 1
```

- **leaderboard_id (required)**

  Type: `integer`

  Description: ID of the leaderboard to map with (must match an entry in leaderboards).

  Example:

```
leaderboard_id: 1
```

- **dataset_split_id (required)**

  Type: integer

  Description: ID of the dataset split to map with (must match an entry in dataset_splits).

  Example:

```
dataset_split_id: 1
```

  To read more about Dataset Splits click here.

Visibility Settings

- **visibility (optional)**

  Type: integer

  Default: 3

  Description: It will set the visibility of the numbers corresponding to metrics for this challenge_phase_split.

  Value:

  - 1: Visible only to challenge hosts
  - 2: Visible to challenge hosts and participants who submitted.
  - 3: Visible to everyone on the leaderboard

  Example:

```
visibility: 2
```

Leaderboard Settings

- **leaderboard_decimal_precision (optional)**

  Type: integer

  Default: 2

  Description: Number of decimal places for leaderboard metrics.

  Example:

```
leaderboard_decimal_precision: 3
```

- **is_leaderboard_order_descending (optional)**

  Type: boolean

  Default: True

  Description: Defines the default leaderboard sorting order. It is useful in cases where you have error as a metric and want to sort the leaderboard in increasing order of error value.

  Value:

  - True: For use cases when higher is better (e.g., accuracy)

- `False`: For use cases when lower is better (e.g., error rate)

Example:

```
is_leaderboard_order_descending: True
```

- **show_execution_time**

  Type: `boolean`

  Description: Defines whether the submission's execution time is displayed on the leaderboard for this split.

  Value:

  - `True`: Display execution time of submissions on leaderboard.
  - `False`: Hide execution time from leaderboard.

  Example:

```
show_execution_time: True
```

- **show_leaderboard_by_latest_submission**

  Type: `boolean`

  Description: Determines whether the leaderboard should be sorted by the latest submission for this split.

  Value:

  - `True`: Leaderboard displays scores based on latest submission.
  - `False`: Leaderboard displays scores based on the best submission by default.

  Example:

```
show_leaderboard_by_latest_submission: True
```

Example:

This is how the challenge phases setup in the challenge configuration YAML file of a sample challenge with all the above fields look like:

```
challenge_phases:
  - id: 1
    name: Dev Phase
    description: templates/challenge_phase_1_description.html
    leaderboard_public: False
    is_public: True
    challenge: 1
    is_active: True
    max_concurrent_submissions_allowed: 3
    allowed_email_ids: []
    disable_logs: False
    is_submission_public: True
    start_date: 2025-07-01 00:00:00
    end_date: 2025-08-31 23:59:59
    test_annotation_file: annotations/test_annotations_devsplit.json
    codename: dev
    max_submissions_per_day: 5
```

(continues on next page)

(continued from previous page)

```
    max_submissions_per_month: 50
    max_submissions: 100
    default_submission_meta_attributes:
      - name: method_name
        is_visible: True
      - name: method_description
        is_visible: True
    submission_meta_attributes:
      - name: TextAttribute
        description: Sample
        type: text
        required: False
      - name: SingleOptionAttribute
        description: Sample
        type: radio
        options: ["A", "B", "C"]
    is_restricted_to_select_one_submission: False
    is_partial_submission_evaluation_enabled: False
    allowed_submission_file_types: ".json, .zip"

  - id: 2
    name: Test Phase
    description: templates/challenge_phase_2_description.html
    leaderboard_public: True
    is_public: True
    challenge: 2
    is_active: True
    max_concurrent_submissions_allowed: 3
    allowed_email_ids: []
    disable_logs: False
    is_submission_public: True
    start_date: 2019-01-01 00:00:00
    end_date: 2099-05-24 23:59:59
    test_annotation_file: annotations/test_annotations_testsplit.json
    codename: test
    max_submissions_per_day: 5
    max_submissions_per_month: 50
    max_submissions: 50
    default_submission_meta_attributes:
      - name: method_name
        is_visible: True
      - name: method_description
        is_visible: True
      - name: project_url
        is_visible: True
      - name: publication_url
        is_visible: True
    submission_meta_attributes:
      - name: TextAttribute
        description: Sample
        type: text
      - name: SingleOptionAttribute
```

(continues on next page)

(continued from previous page)

```
        description: Sample
        type: radio
        options: ["A", "B", "C"]
      - name: MultipleChoiceAttribute
        description: Sample
        type: checkbox
        options: ["alpha", "beta", "gamma"]
      - name: TrueFalseField
        description: Sample
        type: boolean
    is_restricted_to_select_one_submission: False
    is_partial_submission_evaluation_enabled: False

challenge_phase_splits:
  - challenge_phase_id: 1
    leaderboard_id: 1
    dataset_split_id: 1
    visibility: 1
    leaderboard_decimal_precision: 2
    is_leaderboard_order_descending: True
    show_execution_time: True
    show_leaderboard_by_latest_submission: True
  - challenge_phase_id: 2
    leaderboard_id: 1
    dataset_split_id: 1
    visibility: 3
    leaderboard_decimal_precision: 2
    is_leaderboard_order_descending: True
    showeceution_time: False
    show_leaderboard_by_latest_submission: False
  - challenge_phase_id: 2
    leaderboard_id: 1
    dataset_split_id: 2
    visibility: 1
    leaderboard_decimal_precision: 2
    is_leaderboard_order_descending: True
    show_execution_time: True
    show_leaderboard_by_latest_submission: True
```

2.2.3 Dataset Splits

Dataset splits define the subset of test-set on which the submissions will be evaluated on. Generally, most challenges have three splits:

1. **train_split** (Allow participants to make a large number of submissions, let them see how they are doing, and let them overfit)
2. **test_split** (Allow a small number of submissions so that they cannot mimic test_set. Use this split to decide the winners for the challenge)
3. **val_split** (Allow participants to make submissions and evaluate on the validation split)

A dataset split has the following subfields:

- **id (required)**

Type: `integer`

Description: Unique numeric identifier for the dataset split. Used internally to reference this split in phase-split mappings.

Example:

```
id: 1
```

- **name (required)**

  Type: `string`

  Constraints: Must be unique.

  Description: Human-readable name of the dataset split. This will be shown in the EvalAI UI and should clearly describe the split's purpose.

  Example:

```
name: Train Split
```

- **codename (required)**

  Type: `string`

  Constraints: Must be unique and must match the codename used in the evaluation script.

  Description: A unique identifier used to map evaluation results to the correct dataset split. This is critical for EvalAI to interpret the scores returned by your evaluation script.

  Example:

```
codename: train_split
```

Example

Here's how the dataset splits configuration will look like in `challenge_config.yaml` file of a sample challenge:

```
dataset_splits:
  - id: 1
    name: Train Split
    codename: train_split
  - id: 2
    name: Test Split
    codename: test_split
  - id: 3
    name: Validation Split
    codename: val_split
```

2.2.4 YAML Reference

An example of a complete `challenge_config.yaml` file of a sample challenge containing all the fields and subfields discussed till now:

```
title: "Autonomous Driving Lane Detection Challenge"
short_description: "Detect lane boundaries from images in real-time."
description: "templates/description.html"
evaluation_details: "templates/evaluation_details.html"
```

(continues on next page)

(continued from previous page)

```
terms_and_conditions: "templates/terms_and_conditions.html"
image: "images/logo/lane_detection_logo.png"
submission_guidelines: "templates/submission_guidelines.html"
leaderboard_description: "The leaderboard shows the evaluation results of your␣
↪submissions based on accuracy and F1 score. The higher the score, the better your␣
↪model performs."
evaluation_script: "evaluation_script/"
remote_evaluation: false
start_date: "2025-09-01 00:00:00"
end_date: "2025-12-01 23:59:59"
published: false
tags:
  - autonomous-driving
  - lane-detection
  - computer-vision
  - real-time-processing
leaderboard:
  - id: 1
    schema: {
      "labels": ["Accuracy", "F1 Score", "Total"],
      "default_order_by": "Total",
      "metadata": {
        "Accuracy": {
          "sort_ascending": false,
          "description": "Overall accuracy of the model"
        },
        "F1 Score": {
          "sort_ascending": false,
          "description": "Weighted F1 score over all classes"
        },
        "Total": {
          "sort_ascending": false,
          "description": "Combined performance metric"
        }
      }
    }


challenge_phases:
  - id: 1
    name: Dev Phase
    description: templates/challenge_phase_1_description.html
    leaderboard_public: False
    is_public: True
    challenge: 1
    is_active: True
    max_concurrent_submissions_allowed: 3
    allowed_email_ids: []
    disable_logs: False
    is_submission_public: True
    start_date: 2025-07-01 00:00:00
    end_date: 2025-08-31 23:59:59
```

(continues on next page)

(continued from previous page)

```
    test_annotation_file: annotations/test_annotations_devsplit.json
    codename: dev
    max_submissions_per_day: 5
    max_submissions_per_month: 50
    max_submissions: 100
    default_submission_meta_attributes:
      - name: method_name
        is_visible: True
      - name: method_description
        is_visible: True
    submission_meta_attributes:
      - name: TextAttribute
        description: Sample
        type: text
        required: False
      - name: SingleOptionAttribute
        description: Sample
        type: radio
        options: ["A", "B", "C"]
    is_restricted_to_select_one_submission: False
    is_partial_submission_evaluation_enabled: False
    allowed_submission_file_types: ".json, .zip"

  - id: 2
    name: Test Phase
    description: templates/challenge_phase_2_description.html
    leaderboard_public: True
    is_public: True
    challenge: 2
    is_active: True
    max_concurrent_submissions_allowed: 3
    allowed_email_ids: []
    disable_logs: False
    is_submission_public: True
    start_date: 2019-01-01 00:00:00
    end_date: 2099-05-24 23:59:59
    test_annotation_file: annotations/test_annotations_testsplit.json
    codename: test
    max_submissions_per_day: 5
    max_submissions_per_month: 50
    max_submissions: 50
    default_submission_meta_attributes:
      - name: method_name
        is_visible: True
      - name: method_description
        is_visible: True
      - name: project_url
        is_visible: True
      - name: publication_url
        is_visible: True
    submission_meta_attributes:
      - name: TextAttribute
```

(continues on next page)

(continued from previous page)

```
        description: Sample
        type: text
      - name: SingleOptionAttribute
        description: Sample
        type: radio
        options: ["A", "B", "C"]
      - name: MultipleChoiceAttribute
        description: Sample
        type: checkbox
        options: ["alpha", "beta", "gamma"]
      - name: TrueFalseField
        description: Sample
        type: boolean
    is_restricted_to_select_one_submission: False
    is_partial_submission_evaluation_enabled: False


dataset_splits:
    - id: 1
      name: Train Split
      codename: train_split
    - id: 2
      name: Test Split
      codename: test_split
    - id: 3
      name: Validation Split
      codename: val_split


challenge_phase_splits:
  - challenge_phase_id: 1
    leaderboard_id: 1
    dataset_split_id: 1
    visibility: 1
    leaderboard_decimal_precision: 2
    is_leaderboard_order_descending: True
    show_execution_time: True
    show_leaderboard_by_latest_submission: True
  - challenge_phase_id: 2
    leaderboard_id: 1
    dataset_split_id: 1
    visibility: 3
    leaderboard_decimal_precision: 2
    is_leaderboard_order_descending: True
    showeceution_time: False
    show_leaderboard_by_latest_submission: False
  - challenge_phase_id: 2
    leaderboard_id: 1
    dataset_split_id: 2
    visibility: 1
    leaderboard_decimal_precision: 2
    is_leaderboard_order_descending: True
```

(continues on next page)

(continued from previous page)

```
    show_execution_time: True
    show_leaderboard_by_latest_submission: True
```

2.2.5 Editing Challenge Configuration

After a challenge has been created on EvalAI, you may need to update some fields—like changing submission limits, fixing typos, or adjusting dates. This guide explains **which fields can be edited**, **how to do updates**, and **how to test configuration changes locally**.

Editable Fields

All the fields in the challenge configuration YAML can typically be updated, except the following:

`challenge_phase_splits`

`dataset_splits`

In addition to this, the above two fields are **irreversible**, which means, dataset splits or phases cannot be deleted once submissions are made.

How to Edit the fields

All the fields that are editable can be edited by either of the following two ways:

- **Editing directly through EvalAI dashboard**

  Once you create your challenge and push it successfully through the challlenge branch, your challenge will be visible on the EvalAI dashboard at `https://eval.ai`.

  Almost all of the fields, except a few, can be edited directly from the dashboard. Such fields will have an "edit" icon (a small pen icon) appearing beside them, or an "upload" sign in case that is a file upload related field.

- **Editing through YAML file**

  However, there are some editable fields like image, leaderboard public/private, etc., which cannot be edited directly through dashboard, and in such cases you can update them through YAML configuration file.

  Once, you edit the fields in the YAML file, simply push the challenge branch again and once all the checks pass, you should see the updated fields in the dashboard.

Note: If your challenge configuration YAML file is not in sync with the dashboard (e.g., some fields are update directly on dashboard but are still outdated in the YAML file), in such a case, pushing your YAML file to the challenge branch will result in making the dashbord fields same as the ones in YAML file.

Hence, keep your YAML file updated and in sync with the dashboard updates, before pushing the file again.

How to test the changes locally before submission

It's best to test the changes on EvalAI's staging server `https://staging.eval.ai` before pushing your changes to the main production URL i.e., `https://eval.ai`.

To do this follow the steps mentioned below:

1. Use the EvalAI starter repository as template.
2. Generate your github personal acccess token and copy it in clipboard.
3. Add the github personal access token in the forked repository's secrets with the name `AUTH_TOKEN`.
4. Now, go to EvalAI staging server to fetch the following details -

    1. `evalai_user_auth_token` - Go to profile page after logging in and click on `Get your Auth Token` to copy your auth token.
    2. `host_team_pk` - Go to host team page and copy the `ID` for the team you want to use for challenge creation.
    3. `evalai_host_url` - Use `https://staging.eval.ai` for staging server.

5. Create a branch with name `challenge` in the forked repository from the `master` branch. Note: Only changes in `challenge` branch will be synchronized with challenge on EvalAI.
6. Add `evalai_user_auth_token` and `host_team_pk` in `github/host_config.json`.
7. Read *EvalAI challenge creation documentation* to know more about how you want to structure your challenge. Once you are ready, start making changes in the yaml file, HTML templates, evaluation script according to your need.
8. To update the challenge configuration and test it locally on staging server, make changes in the repository and push on `challenge` branch and wait for the build to complete. View the logs of your build.
9. If challenge config contains errors then a `issue` will be opened automatically in the repository with the errors otherwise the challenge will be created on EvalAI Staging server.
10. Go to Hosted Challenges to view your challenge on the staging server.

The challenge will be publicly available once you push it to the main production server i.e., `https://eval.ai` and admin approves the challenge.

For evaluation setup and configuration, checkout the Evaluation section:

- Evaluation

2.3 Evaluation

2.3.1 Evaluation Scripts

Each challenge has an evaluation script, which evaluates the submission of participants and returns the scores which will populate the leaderboard. The logic for evaluating and judging a submission is customizable and varies from challenge to challenge, but the overall structure of evaluation scripts are fixed due to architectural reasons.

Writing an Evaluation Script

Evaluation scripts are required to have an `evaluate()` function. This is the main function, which is used by workers to evaluate the submission messages.

The syntax of evaluate function is:

```
def evaluate(test_annotation_file, user_annotation_file, phase_codename, **kwargs):
    if phase_codename == "dev":
        # Perform evaluation specific to the dev phase
        ...
```

It receives three arguments, namely:

- `test_annotation_file`: It represents the local path to the annotation file for the challenge. This is the file uploaded by the Challenge host while creating a challenge.
- `user_annotation_file`: It represents the local path of the file submitted by the user for a particular challenge phase.
- `phase_codename`: It is the `codename` of the challenge phase from the challenge configuration yaml. This is passed as an argument so that the script can take actions according to the challenge phase.

After reading the files, some custom actions can be performed. This varies per challenge.

The `evaluate()` method also accepts keyword arguments. By default, we provide you metadata of each submission to your challenge which you can use to send notifications to your slack channel or to some other webhook service. Following is an example code showing how to get the submission metadata in your evaluation script and send a slack notification if the accuracy is more than some value `X` (X being 90 in the example given below).

```
def evaluate(test_annotation_file, user_annotation_file, phase_codename, **kwargs):

    submission_metadata = kwargs.get("submission_metadata")
    print submission_metadata

    # Do stuff here
    # Set `score` to 91 as an example

    score = 91
    if score > 90:
        slack_data = kwargs.get("submission_metadata")
        webhook_url = "Your slack webhook url comes here"
        # To know more about slack webhook, checkout this link: https://api.slack.com/
↪incoming-webhooks

        response = requests.post(
            webhook_url,
            data=json.dumps({'text': "*Flag raised for submission:* \n \n" + str(slack_
↪data)}),
            headers={'Content-Type': 'application/json'})

    # Do more stuff here
```

The above example can be modified and used to find if some participant team is cheating or not. There are many more ways for which you can use this metadata.

After all the processing is done, this `evaluate()` should return an output, which is used to populate the leaderboard. The output should be in the following format:

```
output = {}
output['result'] = [
            {
                'train_split': {
                    'Metric1': 123,
                    'Metric2': 123,
                    'Metric3': 123,
                    'Total': 123,
                }
            },
            {
                'test_split': {
                    'Metric1': 123,
                    'Metric2': 123,
                    'Metric3': 123,
                    'Total': 123,
                }
            }
        ]
```

(continues on next page)

(continued from previous page)

```
return output
```

Let's break down what is happening in the above code snippet.

1. `output` should contain a key named `result`, which is a list containing entries per dataset split that is available for the challenge phase in consideration (in the function definition of `evaluate()` shown above, the argument: `phase_codename` will receive the *codename* for the challenge phase against which the submission was made).
2. Each entry in the list should be a dict that has a key with the corresponding dataset split codename (`train_split` and `test_split` for this example).
3. Each of these dataset split dict contains various keys (`Metric1`, `Metric2`, `Metric3`, `Total` in this example), which are then displayed as columns in the leaderboard.

Editing Evaluation Script

Each prediction upload challenge has an evaluation script, which evaluates the submission of participants and returns the scores which will populate the leaderboard. The logic for evaluating a submission is customizable and varies from challenge to challenge.

When setting up a new challenge, hosts need to test multiple versions of evaluation script on EvalAI. To test multiple versions of evaluation script host can update the evaluation script of existing challenge without uploading a whole new challenge configuration.

To edit the evaluation script for existing challenge please follow the following steps:

1. Go to the challenge page

Go to hosted challenges and select the challenge to update evaluation script

2. Navigate to Evaluation criteria tab

Select the Evaluation criteria tab and click on 'upload' button

3. Update the evaluation script

Upload the latest evaluation script and click on 'Submit' button to update the evaluation script

Tada! you have successfully updated the evaluation script for a challenge. The evaluation workers for the challenge will be restarted automatically to pick up the latest evaluation script. Please wait for a minimum of 10 minutes for the workers to restart.

2.3.2 Prediction Upload

2.3.3 Remote Evaluation

Each challenge has an evaluation script, which evaluates the submission of participants and returns the scores which will populate the leaderboard. The logic for evaluating and judging a submission is customizable and varies from challenge to challenge, but the overall structure of evaluation scripts is fixed due to architectural reasons.

Writing Remote Evaluation Script

The starter template for remote challenge evaluation can be found here.

Here are the steps to configure remote evaluation:

1. **Setup Configs**:

   To configure authentication for the challenge set the following environment variables:

   1. AUTH_TOKEN: Go to profile page -> Click on `Get your Auth Token` -> Click on the Copy button. The auth token will get copied to your clipboard.
   2. API_SERVER: Use `https://eval.ai` when setting up challenge on production server. Otherwise, use `https://staging.eval.ai`
   3. QUEUE_NAME: Go to the challenge manage tab to fetch the challenge queue name.
   4. CHALLENGE_PK: Go to the challenge manage tab to fetch the challenge primary key.
   5. SAVE_DIR: (Optional) Path to submission data download location.

2. **Write `evaluate` method**: Evaluation scripts are required to have an `evaluate()` function. This is the main function, which is used by workers to evaluate the submission messages.

   The syntax of evaluate function for a remote challenge is:

```
def evaluate(user_submission_file, phase_codename, test_annotation_file = None,␣
↪**kwargs)
    pass
```

   It receives three arguments, namely:

   - `user_annotation_file`: It represents the local path of the file submitted by the user for a particular challenge phase.
   - `phase_codename`: It is the `codename` of the challenge phase from the challenge configuration yaml. This is passed as an argument so that the script can take actions according to the challenge phase.
   - `test_annotation_file`: It represents the local path to the annotation file for the challenge. This is the file uploaded by the Challenge host while creating a challenge.

   You may pass the `test_annotation_file` as default argument or choose to pass separately in the `main.py` depending on the case. The `phase_codename` is passed automatically but is left as an argument to allow customization.

   After reading the files, some custom actions can be performed. This varies per challenge.

   The `evaluate()` method also accepts keyword arguments.

   IMPORTANT : If the `evaluate()` method fails due to any reason or there is a problem with the submission, please ensure to raise an `Exception` with an appropriate message.

2.3.4 Metric Leaderboards

2.4 Templates

2.4.1 Challenge Configuration Examples

This guide provides a set of fully functional examples of `challenge_config.yaml` file for common challenge setups on EvalAI. Each example illustrates a different real-world use case—ranging from single-phase challenges to multi-phase, multi-leaderboard competitions. These templates are designed to help organizers get started quickly, and understand how various components of the configuration work together.

Before using any of the following templates as your challenge configuration, make sure all the mentioned files exist in the respective paths as mentioned in different fields of the configuration. (e.g., `test_annotation_file`, `description`, etc.)

Example 1: One Challenge, One Phase, One Leaderboard, One Phase Split

example_1_config.yaml:

```yaml
title: "Image Classification Challenge"
short_description: "Classify images into categories."
description: "templates/description.html"
evaluation_details: "templates/evaluation_details.html"
terms_and_conditions: "templates/terms_and_conditions.html"
image: "logo.jpg"
submission_guidelines: "templates/submission_guidelines.html"
leaderboard_description: "The leaderboard shows the evaluation results of your␣
↪submissions based on accuracy. Higher is better."
evaluation_script: "evaluation_script.zip"
remote_evaluation: false
start_date: "2025-07-01 00:00:00"
end_date: "2025-12-01 23:59:59"
published: false
tags:
  - image-classification
  - supervised-learning
  - computer-vision

leaderboard:
  - id: 1
    schema: {
      "labels": ["Accuracy"],
      "default_order_by": "Accuracy",
      "metadata": {
        "Accuracy": {
          "sort_ascending": false,
          "description": "Classification accuracy on the test set"
        }
      }
    }

challenge_phases:
  - id: 1
    name: Dev Phase
    description: templates/challenge_phase_1_description.html
    leaderboard_public: True
    is_public: True
    challenge: 1
    is_active: True
    max_concurrent_submissions_allowed: 3
    allowed_email_ids: []
    disable_logs: False
    is_submission_public: True
    start_date: 2025-07-01 00:00:00
    end_date: 2025-10-01 23:59:59
```

(continues on next page)

(continued from previous page)

```
    test_annotation_file: annotations/test_annotations_devsplit.json
    codename: dev
    max_submissions_per_day: 5
    max_submissions_per_month: 50
    max_submissions: 100
    default_submission_meta_attributes:
      - name: method_name
        is_visible: True
      - name: method_description
        is_visible: True
    submission_meta_attributes:
      - name: description
        description: Describe your classification method
        type: text
        required: True
      - name: model_type
        description: Select model type
        type: radio
        options: ["CNN", "Transformer", "Other"]
      - name: pre_trained
        description: Is the model pre-trained?
        type: boolean
    is_restricted_to_select_one_submission: False
    is_partial_submission_evaluation_enabled: False
    allowed_submission_file_types: ".json, .zip"

dataset_splits:
  - id: 1
    name: Test Split
    codename: test_split

challenge_phase_splits:
  - challenge_phase_id: 1
    leaderboard_id: 1
    dataset_split_id: 1
    visibility: 1
    leaderboard_decimal_precision: 2
    is_leaderboard_order_descending: True
    show_execution_time: True
    show_leaderboard_by_latest_submission: True
```

Example 2: One Challenge, One Phase, Two Leaderboards, Two Phase Splits

example_2_config.yaml:

```
title: "Simple QA Challenge"
short_description: "Answer simple factual questions."
description: "templates/description.html"
evaluation_details: "templates/evaluation_details.html"
terms_and_conditions: "templates/terms_and_conditions.html"
image: "logo.jpg"
submission_guidelines: "templates/submission_guidelines.html"
```

(continues on next page)

(continued from previous page)

```yaml
leaderboard_description: "The leaderboard tracks performance based on accuracy and speed.
↪"
evaluation_script: "evaluation_script.zip"
remote_evaluation: false
start_date: "2025-10-01 00:00:00"
end_date: "2026-01-31 23:59:59"
published: false
tags:
  - question-answering
  - qa
  - text

leaderboard:
  - id: 1
    schema: {
      "labels": ["Correct Answers"],
      "default_order_by": "Correct Answers",
      "metadata": {
        "Correct Answers": {
          "sort_ascending": false,
          "description": "Total number of correct answers"
        }
      }
    }

  - id: 2
    schema: {
      "labels": ["Average Response Time"],
      "default_order_by": "Average Response Time",
      "metadata": {
        "Average Response Time": {
          "sort_ascending": true,
          "description": "Average time taken to answer a question"
        }
      }
    }

challenge_phases:
  - id: 1
    name: QA Phase
    description: templates/challenge_phase_1_description.html
    leaderboard_public: true
    is_public: true
    challenge: 1
    is_active: true
    max_concurrent_submissions_allowed: 3
    allowed_email_ids: []
    disable_logs: false
    is_submission_public: true
    start_date: 2025-10-01 00:00:00
    end_date: 2026-01-31 23:59:59
    test_annotation_file: annotations/test_annotations_devsplit.json
```

(continues on next page)

(continued from previous page)

```
    codename: qa
    max_submissions_per_day: 5
    max_submissions_per_month: 50
    max_submissions: 100
    default_submission_meta_attributes:
      - name: method_name
        is_visible: true
      - name: method_description
        is_visible: true
    submission_meta_attributes:
      - name: System Type
        description: Choose system type
        type: radio
        options: ["Rule-Based", "ML-Based"]
      - name: Open Source
        description: Is it open source?
        type: boolean
        required: false
    is_restricted_to_select_one_submission: false
    is_partial_submission_evaluation_enabled: false
    allowed_submission_file_types: ".json, .zip"

dataset_splits:
  - id: 1
    name: Accuracy Split
    codename: accuracy_split
  - id: 2
    name: Speed Split
    codename: speed_split

challenge_phase_splits:
  - challenge_phase_id: 1
    leaderboard_id: 1
    dataset_split_id: 1
    visibility: 1
    leaderboard_decimal_precision: 0
    is_leaderboard_order_descending: true
    show_execution_time: true
    show_leaderboard_by_latest_submission: false

  - challenge_phase_id: 1
    leaderboard_id: 2
    dataset_split_id: 2
    visibility: 1
    leaderboard_decimal_precision: 2
    is_leaderboard_order_descending: true
    show_execution_time: true
    show_leaderboard_by_latest_submission: false
```

Example 3: One Challenge, Two Phases, One Leaderboard, Two Phase Splits

example_3_config.yaml:

```yaml
title: "Object Detection Challenge"
short_description: "Detect objects in images."
description: "templates/description.html"
evaluation_details: "templates/evaluation_details.html"
terms_and_conditions: "templates/terms_and_conditions.html"
image: "logo.jpg"
submission_guidelines: "templates/submission_guidelines.html"
leaderboard_description: "Leaderboard evaluates detection performance using correct,␣
↪missed, and combined scores."
evaluation_script: "evaluation_script.zip"
remote_evaluation: false
start_date: "2025-08-01 00:00:00"
end_date: "2026-01-01 23:59:59"
published: false
tags:
  - object-detection
  - computer-vision

leaderboard:
  - id: 1
    schema: {
      "labels": ["Correct Detections", "Missed Detections", "Overall Score"],
      "default_order_by": "Overall Score",
      "metadata": {
        "Correct Detections": {
          "sort_ascending": false,
          "description": "Number of correct object predictions"
        },
        "Missed Detections": {
          "sort_ascending": true,
          "description": "Number of objects the model failed to detect"
        },
        "Overall Score": {
          "sort_ascending": false,
          "description": "Combined score of accuracy and completeness"
        }
      }
    }

challenge_phases:
  - id: 1
    name: Dev Phase
    description: templates/challenge_phase_1_description.html
    leaderboard_public: true
    is_public: true
    challenge: 1
    is_active: true
    max_concurrent_submissions_allowed: 3
    allowed_email_ids: []
    disable_logs: false
```

(continues on next page)

(continued from previous page)

```
    is_submission_public: true
    start_date: 2025-08-01 00:00:00
    end_date: 2025-10-15 23:59:59
    test_annotation_file: annotations/test_annotations_devsplit.json
    codename: dev
    max_submissions_per_day: 5
    max_submissions_per_month: 50
    max_submissions: 100
    default_submission_meta_attributes:
      - name: method_name
        is_visible: true
      - name: method_description
        is_visible: true
    submission_meta_attributes:
      - name: framework
        description: Framework used
        type: radio
        options: ["PyTorch", "TensorFlow", "Other"]
        required: false
      - name: open_source
        description: Is your code open source?
        type: boolean
        required: false
    is_restricted_to_select_one_submission: false
    is_partial_submission_evaluation_enabled: false
    allowed_submission_file_types: ".json, .zip"

  - id: 2
    name: Test Phase
    description: templates/challenge_phase_2_description.html
    leaderboard_public: true
    is_public: true
    challenge: 1
    is_active: false
    max_concurrent_submissions_allowed: 3
    allowed_email_ids: []
    disable_logs: false
    is_submission_public: true
    start_date: 2025-10-16 00:00:00
    end_date: 2026-01-01 23:59:59
    test_annotation_file: annotations/test_annotations_testsplit.json
    codename: test
    max_submissions_per_day: 5
    max_submissions_per_month: 50
    max_submissions: 100
    default_submission_meta_attributes:
      - name: method_name
        is_visible: true
      - name: method_description
        is_visible: true
    submission_meta_attributes:
      - name: framework
```

(continues on next page)

(continued from previous page)

```
        description: Framework used
        type: radio
        options: ["PyTorch", "TensorFlow", "Other"]
        required: false
      - name: open_source
        description: Is your code open source?
        type: boolean
        required: false
    is_restricted_to_select_one_submission: false
    is_partial_submission_evaluation_enabled: false
    allowed_submission_file_types: ".json, .zip"

dataset_splits:
  - id: 1
    name: Dev Split
    codename: dev_split
  - id: 2
    name: Test Split
    codename: test_split

challenge_phase_splits:
  - challenge_phase_id: 1
    leaderboard_id: 1
    dataset_split_id: 1
    visibility: 1
    leaderboard_decimal_precision: 2
    is_leaderboard_order_descending: true
    show_execution_time: true
    show_leaderboard_by_latest_submission: true

  - challenge_phase_id: 2
    leaderboard_id: 1
    dataset_split_id: 2
    visibility: 1
    leaderboard_decimal_precision: 2
    is_leaderboard_order_descending: true
    show_execution_time: true
    show_leaderboard_by_latest_submission: true
```

Example 4: One Challenge, Three Phases, Two Leaderboards, Four Phase Splits

example_4_config.yaml:

```
title: "Multi-Stage NLP Challenge"
short_description: "NLP challenge with public/private testing."
description: "templates/description.html"
evaluation_details: "templates/evaluation_details.html"
terms_and_conditions: "templates/terms_and_conditions.html"
image: "logo.jpg"
submission_guidelines: "templates/submission_guidelines.html"
leaderboard_description: "Leaderboard evaluates both match accuracy and mismatch rate␣
↪across multiple stages."
```

(continues on next page)

(continued from previous page)

```yaml
evaluation_script: "evaluation_script.zip"
remote_evaluation: false
start_date: "2025-09-01 00:00:00"
end_date: "2026-03-01 23:59:59"
published: false
tags:
  - nlp
  - machine-learning

leaderboard:
  - id: 1
    schema: {
      "labels": ["Match Accuracy"],
      "default_order_by": "Match Accuracy",
      "metadata": {
        "Match Accuracy": {
          "sort_ascending": false,
          "description": "How accurately the model predicts matched pairs"
        }
      }
    }

  - id: 2
    schema: {
      "labels": ["Mismatch Rate"],
      "default_order_by": "Mismatch Rate",
      "metadata": {
        "Mismatch Rate": {
          "sort_ascending": true,
          "description": "How often the model incorrectly identifies non-matches"
        }
      }
    }

challenge_phases:
  - id: 1
    name: Dev Phase
    description: templates/challenge_phase_1_description.html
    leaderboard_public: true
    is_public: true
    challenge: 1
    is_active: true
    max_concurrent_submissions_allowed: 3
    allowed_email_ids: []
    disable_logs: false
    is_submission_public: true
    start_date: 2025-09-01 00:00:00
    end_date: 2025-10-15 23:59:59
    test_annotation_file: annotations/test_annotations_devsplit.json
    codename: dev
    max_submissions_per_day: 5
    max_submissions_per_month: 50
```

(continues on next page)

(continued from previous page)

```
    max_submissions: 100
    default_submission_meta_attributes:
      - name: method_name
        is_visible: true
      - name: method_description
        is_visible: true
    submission_meta_attributes:
      - name: pretraining
        description: Was the model pretrained?
        type: radio
        options: ["Yes", "No"]
        required: false
      - name: public_code
        description: Is your code publicly available?
        type: boolean
        required: false
    is_restricted_to_select_one_submission: false
    is_partial_submission_evaluation_enabled: false
    allowed_submission_file_types: ".json, .zip"

  - id: 2
    name: Public Test
    description: templates/challenge_phase_2_description.html
    leaderboard_public: true
    is_public: true
    challenge: 1
    is_active: false
    max_concurrent_submissions_allowed: 3
    allowed_email_ids: []
    disable_logs: false
    is_submission_public: true
    start_date: 2025-10-16 00:00:00
    end_date: 2025-12-01 23:59:59
    test_annotation_file: annotations/test_annotations_public_split.json
    codename: test-public
    max_submissions_per_day: 5
    max_submissions_per_month: 50
    max_submissions: 100
    default_submission_meta_attributes:
      - name: method_name
        is_visible: true
      - name: method_description
        is_visible: true
    submission_meta_attributes:
      - name: pretraining
        description: Was the model pretrained?
        type: radio
        options: ["Yes", "No"]
        required: false
      - name: public_code
        description: Is your code publicly available?
        type: boolean
```

(continues on next page)

(continued from previous page)

```
        required: false
    is_restricted_to_select_one_submission: false
    is_partial_submission_evaluation_enabled: false
    allowed_submission_file_types: ".json, .zip"

  - id: 3
    name: Private Test
    description: templates/challenge_phase_3_description.html
    leaderboard_public: true
    is_public: false
    challenge: 1
    is_active: false
    max_concurrent_submissions_allowed: 3
    allowed_email_ids: []
    disable_logs: false
    is_submission_public: true
    start_date: 2025-12-02 00:00:00
    end_date: 2026-03-01 23:59:59
    test_annotation_file: annotations/test_annotations_private_split.json
    codename: test-private
    max_submissions_per_day: 5
    max_submissions_per_month: 50
    max_submissions: 100
    default_submission_meta_attributes:
      - name: method_name
        is_visible: true
      - name: method_description
        is_visible: true
    submission_meta_attributes:
      - name: pretraining
        description: Was the model pretrained?
        type: radio
        options: ["Yes", "No"]
        required: false
      - name: public_code
        description: Is your code publicly available?
        type: boolean
        required: false
    is_restricted_to_select_one_submission: false
    is_partial_submission_evaluation_enabled: false
    allowed_submission_file_types: ".json, .zip"

dataset_splits:
  - id: 1
    name: Dev Split-1
    codename: dev_split
  - id: 2
    name: Public Test
    codename: test_pub_split
  - id: 3
    name: Private Test
    codename: test_priv_split
```

(continues on next page)

(continued from previous page)

```
challenge_phase_splits:
  - challenge_phase_id: 1
    leaderboard_id: 1
    dataset_split_id: 1
    visibility: 1
    leaderboard_decimal_precision: 2
    is_leaderboard_order_descending: true
    show_execution_time: true
    show_leaderboard_by_latest_submission: true

  - challenge_phase_id: 1
    leaderboard_id: 2
    dataset_split_id: 1
    visibility: 1
    leaderboard_decimal_precision: 2
    is_leaderboard_order_descending: true
    show_execution_time: true
    show_leaderboard_by_latest_submission: true

  - challenge_phase_id: 2
    leaderboard_id: 2
    dataset_split_id: 2
    visibility: 1
    leaderboard_decimal_precision: 2
    is_leaderboard_order_descending: true
    show_execution_time: true
    show_leaderboard_by_latest_submission: true

  - challenge_phase_id: 3
    leaderboard_id: 2
    dataset_split_id: 3
    visibility: 1
    leaderboard_decimal_precision: 2
    is_leaderboard_order_descending: true
    show_execution_time: true
    show_leaderboard_by_latest_submission: true
```

2.4.2 HTML Templates

2.4.3 Submission Guidelines

CHAPTER

THREE

FOR PARTICIPANTS

3.1 Participants Guide

3.1.1 Getting Started

3.1.2 Team Management

3.1.3 Participate

You have to create an account on EvalAI and a participant team in order to participate in a challenge.

If you are already familiar with the flow of EvalAI, you may want to skip this section else please follow the following steps to participate in a challenge (VQA Challenge 2017 in this example):

1. Visit eval.ai

Open EvalAI website.

2. Sign up or Log in

Sign Up and fill in your credentials or log in if you have already registered.

After signing up you would be on the dashboard page.

3. Choose challenge

Then, go to challenges section and choose an active challenge.

4. Challenge Page

After reading the challenge instructions on the challenge page, you can participate in the challenge.

5. Create Participant Team

Create a participant team if there isn't any or you can select from the existing ones.

Click on 'Participate' tab after selecting a team.

Tada! you have successfully participated in a challenge.

3.2 Submissions

3.2.1 Submissions Types

3.2.2 Prediction Upload

3.2.3 Submission Status

How is a submission processed?

We are using REST APIs along with Queue based architecture to process submissions. When a participant makes a submission for a challenge, a REST API with URL pattern `jobs:challenge_submission` is called. This API does the task of creating a new entry for submission model and then publishes a message to exchange `evalai_submissions` with a routing key of `submission.*.*`.

```
User Submission  --> API  --> Publish  --> SQS Queue  --> Submission
                              message                      worker(s)
```

Exchange receives the message and then routes it to the queue `submission_task_queue`. At the end of `submission_task_queue` are workers (scripts/workers/submission_worker.py) which processes the submission message.

The worker can be run with

```
# assuming the current working directory is where manage.py lives
python scripts/workers/submission_worker.py
```

How does submission worker function?

Submission worker is a python script which mostly runs as a daemon on a production server and simply acts as a python process in a development environment. To run submission worker in a development environment:

```
python scripts/workers/submission_worker.py
```

Before a worker fully starts, it does the following actions:

- Creates a new temporary directory for storing all its data files.
- Fetches the list of active challenges from the database. Active challenges are published challenges whose start date is less than present time and end date greater than present time. It loads all the challenge evaluation scripts in a variable called `EVALUATION_SCRIPTS`, with the challenge id as its key. The maps looks like this:

  ```
  EVALUATION_SCRIPTS = {
      <challenge_pk> : <evalutaion_script_loaded_as_module>,
      ....
  }
  ```

- Creates a connection with SQS queue by using the AWS supplied credentials as environment variables.
- After the connection is successfully created, creates an exchange with the name `evalai_submissions` and two queues, one for processing submission message namely `submission_task_queue`, and other for getting add challenge message.
- `submission_task_queue` is then bound with the routing key of `submission.*.*` and add challenge message queue is bound with a key of `challenge.add.*` Whenever a queue is bound to a exchange with any key, it will route the message to the corresponding queue as soon as the exchange receives a message with a key.
- Binding to any queue is also accompanied with a callback which basically takes a function as an argument. This function specifies what should be done when the queue receives a message.

e.g. `submission_task_queue` is using `process_submission_callback` as a function, which means that when a message is received in the queue, `process_submission_callback` will be called with the message passed as an argument.

Expressing it informally it will be something like

> *Queue*: Hey *Exchange*, I am `submission_task_queue`. I will be listening to messages from you on binding key of `submission.*.*`
>
> *Exchange*: Hey *Queue*, Sure! When I receive a message with a routing key of `submission.*.*`, I will give it to you
>
> *Queue*: Thanks a lot.
>
> *Queue*: Hey *Worker*, Just for the record, when I receive a new message for submission, I want `process_submission_callback` to be called. Can you please make a note of it?
>
> *Worker*: Sure *Queue*, I will invoke `process_submission_callback` whenever you receive a new message.

When a worker starts, it fetches active challenges from the database and then loads all the challenge evaluation scripts in a variable called `EVALUATION_SCRIPTS`, with challenge id as its key. The map would look like

```
EVALUATION_SCRIPTS = {
    <challenge_pk> : <evalutaion_script_loaded_as_module>,
    ....
}
```

After the challenges are successfully loaded, it creates a connection with the SQS queue `evalai_submission_queue` and listens to it for new submissions.

How is submission made?

When the user makes a submission on the frontend, the following actions happen sequentially

- As soon as the user submits a submission, a REST API with the URL pattern `jobs:challenge_submission` is called.
- This API fetches the challenge and its corresponding challenge phase.
- This API then checks if the challenge is active and challenge phase is public.
- It fetches the participant team's ID and its corresponding object.
- After all these checks are complete, a submission object is saved. The saved submission object includes **participant team id** and **challenge phase id** and **username** of the participant creating it.
- At the end, a submission message is published to exchange `evalai_submissions` with a routing key of `submission.*.*`.

Format of submission messages

The format of the message is

```
{
    "challenge_id": <challenge_pk_here>,
    "phase_id": <challenge_phase_pk_here>,
    "submission_id": <submission_pk_here>
}
```

This message is published with a routing key of `submission.*.*`

How workers process submission message

Upon receiving a message from `submission_task_queue` with a binding key of `submission.*.*`, `process_submission_callback` is called. This function does the following:

- It fetches the challenge phase and submission object from the database using the challenge phase id and submission id received in the message.
- It then downloads the required files like input_file, etc. for submission in its computation directory.
- After this, the submission is run. Submission is initially marked in **RUNNING** state. The `evaluate` function of `EVALUATION_SCRIPTS` map with key of the challenge id is called. The `evaluate` function takes in the annotation file path, the user annotation file path, and the challenge phase's code name as arguments. Running a submission involves temporarily updating `stderr` and `stdout` to different locations other than standard locations. This is done so as to capture the output and any errors produced when running the submission.
- The output from the `evaluate` function is stored in a variable called `submission_output`. Currently, the only way to check for the occurrence of an error is to check if the key `result` exists in `submission_output`.
  - If the key does not exist, then the submission is marked as **FAILED**.
  - If the key exists, then the variable `submission_output` is parsed and `DataSetSplit` objects are created. LeaderBoardData objects are also created (in bulk) with the required parameters. Finally, the submission is marked as **FINISHED**.
- The value in the temporarily updated `stderr` and `stdout` are stored in files named `stderr.txt` and `stdout.txt` which are then stored in the submission instance.
- Finally, the temporary computation directory allocated for this submission is removed.

Notes

- REST API with url pattern `jobs:challenge_submission`. Here *jobs* is application namespace and *challenge_submission* is instance namespace. You can read more about url namespace

3.2.4 Troubleshooting

3.3 CLI Tools

3.3.1 EvalAI CLI

3.3.2 CLI Commands

3.3.3 CLI Troubleshooting

3.4 Visibility

3.4.1 Public Submissions

Let's assume that you want to make your latest submission public in `William Challenge`.

1. Go to `My Submissions` Tab of the challenge page, select the phase and scroll horizontally.
2. Now, to make the first submission public, click on the checkbox under the column `Show on Leaderboard`. It will turn into a green checkmark.

3.4.2 Private Submissions

Let's assume that you want to make your latest submission private in `William Challenge`.

1. Go to `My Submissions` Tab of the challenge page, select the phase and scroll horizontally.
2. Now, to make the first submission private, click on the green checkmark under the column `Show on Leaderboard`. It will turn into a black checkbox.

If you face any issues, please feel free to create an issue on our GitHub Issues Page.

3.4.3 Baseline Submissions

CHAPTER

FOUR

DEVELOPMENT

4.1 Architecture

4.1.1 System Architecture

EvalAI helps researchers, students, and data scientists to create, collaborate, and participate in various AI challenges organized around the globe. To achieve this, we leverage some of the best open source tools and technologies.

Django

Django is the heart of the application, which powers our backend. We use Django version 1.11.23.

Django Rest Framework

We use Django Rest Framework for writing and providing REST APIs. Its permission and serializers have helped write a maintainable codebase.

Amazon SQS

We currently use Amazon SQS for queueing submission messages which are then later on processed by a Python worker.

PostgreSQL

PostgreSQL is used as our primary datastore. All our tables currently reside in a single database named `evalai`

Angular JS

Angular JS is a well-known framework that powers our frontend.

4.1.2 Architectural Decisions

This is a collection of records for architecturally significant decisions.

URL Patterns

We follow a very basic, yet strong convention for URLs, so that our rest APIs are properly namespaced. First of all, we rely heavily on HTTP verbs to perform **CRUD** actions.

For example, to perform **CRUD** operation on *Challenge Host Model*, the following URL patterns will be used.

- `GET /hosts/challenge_host_team` - Retrieves a list of challenge host teams
- `POST /hosts/challenge_host_team` - Creates a new challenge host team

- `GET /hosts/challenge_host_team/<challenge_host_team_id>` - Retrieves a specific challenge host team
- `PUT /hosts/challenge_host_team/<challenge_host_team_id>` - Updates a specific challenge host team
- `PATCH /hosts/challenge_host_team/<challenge_host_team_id>` - Partially updates a specific challenge host team
- `DELETE /hosts/challenge_host_team/<challenge_host_team_id>` - Deletes a specific challenge host team

Also, we have namespaced the URL patterns on a per-app basis, so URLs for *Challenge Host Model*, which is in the *hosts* app, will be

```
/hosts/challenge_host_team
```

This way, one can easily identify where a particular API is located.

We use underscore _ in URL patterns.

Processing submission messages asynchronously

When a submission message is made, a REST API is called which saves the data related to the submission in the database. A submission involves the processing and evaluation of `input_file`. This file is used to evaluate the submission and then decide the status of the submission, whether it is *FINISHED* or *FAILED*.

One way to process the submission is to evaluate it as soon as it is made, hence blocking the participant's request. Blocking the request here means to send the response to the participant only when the submission has been made and its output is known. This would work fine if the number of the submissions made is very low, but this is not the case.

Hence we decided to process and evaluate submission message in an asynchronous manner. To process the messages this way, we need to change our architecture a bit and add a Message Framework, along with a worker so that it can process the message.

Out of all the awesome messaging frameworks available, we have chosen Amazon Simple Queue Service (SQS) because it can support decoupled environments. It allows developers to focus on application development, rather than creating their own sophisticated message-based applications. It also eliminates queuing management tasks, such as storage. SQS also works with AWS resources, so you can use it to make reliable and scalable applications on top of an AWS infrastructure.

For the worker, we went ahead with a normal python worker, which simply runs a process and loads all the required data in its memory. As soon as the worker starts, it listens on a SQS queue named `evalai_submission_queue` for new submission messages.

Submission Worker

The submission worker is responsible for processing submission messages. It listens on a queue named `evalai_submission_queue`, and on receiving a message for a submission, it processes and evaluates the submission.

One of the major design changes that we decided to implement in the submission worker was to load all the data related to the challenge in the worker's memory, instead of fetching it every time a new submission message arrives. So the worker, when starting, fetches the list of active challenges from the database and then loads it into memory by maintaining the map `EVALUATION_SCRIPTS` on challenge id. This was actually a major performance improvement.

Another major design change that we incorporated here was to dynamically import the challenge module and to load it in the map instead of invoking a new python process every time a submission message arrives. So now whenever a new message for a submission is received, we already have its corresponding challenge module being loaded in a map called `EVALUATION_SCRIPTS`, and we just need to call

```
EVALUATION_SCRIPTS[challenge_id].evaluate(*params)
```

This was again a major performance improvement, which saved us from the task of invoking and managing Python processes to evaluate submission messages. Also, invoking a new python process every time for a new submission would have been really slow.

4.1.3 Directory Structure

Django apps

EvalAI is a Django-based application, hence it leverages the concept of Django apps to properly namespace the functionalities. All the apps can be found in the `apps` directory situated in the root folder.

Some important apps along with their main uses are:

- **Challenges**

This app handles all the workflow related to creating, modifying, and deleting challenges.

- **Hosts**

This app is responsible for providing functionalities to the challenge hosts/organizers.

- **Participants**

This app serves users who want to take part in any challenge. It contains code for creating a Participant Team, through which they can participate in any challenge.

- **Jobs**

One of the most important apps, responsible for processing and evaluating submissions made by participants. It contains code for creating a submission, changing the visibility of the submission and populating the leaderboard for any challenge.

- **Web**

This app serves some basic functionalities like providing support for contact us or adding a new contributor to the team, etc.

- **Accounts**

As the name indicates, this app deals with storing and managing data related to user accounts.

- **Base**

A placeholder app which contains the code that is used across various other apps.

Settings

Settings are used across the backend codebase by Django to provide config values on a per-environment basis. Currently, the following settings are available:

- **dev**

Used in development environment

- **testing**

Used whenever test cases are run

- **staging**

Used on staging server

- **production**

Used on production server

URLs

The base URLs for the project are present in `evalai/urls.py`. This file includes URLs of various applications, which are also namespaced by the app name. So URLs for the `challenges` app will have its app namespace in the URL as `challenges`. This actually helps us separate our API based on the app.

Frontend

The whole codebase for the frontend resides in a folder named `frontend` in the root directory

Scripts

Scripts contain various helper scripts, utilities, python workers. It contains the following folders:

- **migration**

Contains some of the scripts which are used for one-time migration or formatting of data.

- **tools**

A folder for storing helper scripts, e.g. a script to fetch pull request

- **workers**

One of the main directories, which contains the code for submission worker. Submission worker is a normal python worker which is responsible for processing and evaluating submission of a user. The command to start a worker is:

```
python scripts/workers/submission_worker.py
```

Test Suite

All of the codebase related to testing resides in `tests` folder in the root directory. In this directory, tests are namespaced according to the app, e.g. tests for `challenges` app lives in a folder named `challenges`.

Management Commands

To perform certain actions like seeding the database, we use Django management commands. Since the management commands are common throughout the project, they are present in `base` application directory. At the moment, the only management command is `seed`, which is used to populate the database with some random values. The command can be invoked by calling

```
python manage.py seed
```

4.1.4 Technology Stack

4.2 Contributing

4.2.1 Contribution Guide

Thank you for your interest in contributing to EvalAI! Here are a few pointers about how you can help.

Setting things up

To set up the development environment, follow the instructions in README.

Finding something to work on

The issue tracker of EvalAI a good place to start. If you find something that interests you, comment on the thread and we'll help get you started.

Alternatively, if you come across a new bug on the site, please file a new issue and comment if you would like to be assigned. The existing issues are tagged with one or more labels, based on the part of the website it touches, its importance etc., that can help you in selecting one.

If neither of these seem appealing, please post on our channel and we will help find you something else to work on.

Instructions to submit code

Before you submit code, please talk to us via the issue tracker so we know you are working on it.

Our central development branch is development. Coding is done on feature branches based off of development and merged into it once stable and reviewed. To submit code, follow these steps:

1. Create a new branch off of development. Select a descriptive branch name.

   ```
   git remote add upstream git@github.com:Cloud-CV/EvalAI.git
   git fetch upstream
   git checkout master
   git merge upstream/master
   git checkout -b your-branch-name
   ```

   We highly encourage using black to format your code. It sticks to PEP8 for the most part and is in line with the rest of the repo. We have already set up pre-commit hooks to run black and flake8. To activate the hooks, you just need to run the following comamnd once:

   ```
   pre-commit install
   ```

2. Commit and push code to your branch:

   - Commits should be self-contained and contain a descriptive commit message.

     #### Rules for a great git commit message style

     - Separate subject from body with a blank line
     - Do not end the subject line with a period
     - Capitalize the subject line and each paragraph
     - Use the imperative mood in the subject line
     - Wrap lines at 72 characters
     - Use the body to explain what and why you have done something. In most cases, you can leave out details about how a change has been made.

     #### Example for a commit message

     ```
     Subject of the commit message
     ```

(continues on next page)

(continued from previous page)

```
Body of the commit message...
....
```

- Please make sure your code is well-formatted and adheres to PEP8 conventions (for Python) and the airbnb style guide (for JavaScript). For others (Lua, prototxt etc.) please ensure that the code is well-formatted and the style consistent.
- Please ensure that your code is well tested.
- We highly encourage to use `autopep8` to follow the PEP8 styling. Run the following command before creating the pull request:

```
autopep8 --in-place --exclude env,docs --recursive .
git commit -a -m "{{commit_message}}"
git push origin {{branch_name}}
```

- Also, For Pretifying the Frontend Code Use `HTML/JS/CSS Pretifier`.
- For installing the Sublime Package Control Manager in Sublime-Text Editor use this link. Also, If Sublime Package Control Manager is installed then install `HTML/JS/CSS Pretifier`.

3. Once the code is pushed, create a pull request:
   - On your GitHub fork, select your branch and click "New pull request". Select "master" as the base branch and your branch in the "compare" dropdown. If the code is mergeable (you get a message saying "Able to merge"), go ahead and create the pull request.
   - Check back after some time to see if the Travis checks have passed, if not you should click on "Details" link on your PR thread at the right of "The Travis CI build failed", which will take you to the dashboard for your PR. You will see what failed / stalled, and will need to resolve them.
   - If your checks have passed, your PR will be assigned a reviewer who will review your code and provide comments. Please address each review comment by pushing new commits to the same branch (the PR will automatically update, so you don't need to submit a new one). Once you are done, comment below each review comment marking it as "Done". Feel free to use the thread to have a discussion about comments that you don't understand completely or don't agree with.
   - Once all comments are addressed, the maintainer will approve the PR.
4. Once you get reviewed by a mentor and done with all the required changes, squash all the commits:

```
git checkout <branch_name>
git rebase -i HEAD~N (N is the number of commits to be squashed)
```

   - Then a screen will appear with all N commits having "pick" written in front of every commit.Change pick to s for the last N-1 commits and let it be pick for the first one.
   - Press esc button and type ":wq" to save the change and close the screen. Now a new screen will appear asking you to change commit message. Change it accordingly and save it. git push origin <branch_name> –force
   - For further query regarding rebasing, visit https://github.com/todotxt/todo.txt-android/wiki/Squash-All-Commits-Related-to-a-Single-Issue-into-a-Single-Commit
   - Once rebasing is done, the reviewer will approve and merge the PR.

Congratulations, you have successfully contributed to Project EvalAI!

4.2.2 Pull Requests

Contributing to EvalAI is really easy. Just follow these steps to get started.

Step 1: Fork

1. Fork the EvalAI repository from the repository.

Step 2: Selecting an issue

1. Select a suitable issue that will be easy for you to fix. Moreover, you can also take the issues based on their labels. All the issues are labelled according to its difficulty.
2. After selecting an issue, ask the maintainers of the project to assign it to you and they will assign it based on its availability.
3. Once it gets assigned, create a branch from your fork's updated master branch using the following command: `git checkout -b branch_name`
4. Start working on the issue.

Step 3: Committing Your Changes

1. After making the changes, you need to add your files to your local git repository.
2. To add your files, use the following commands:

- To add only modified files, use `git add -u`
- To add a new file, use `git add file_path_from_local_git_repository`
- To add all files, use `git add .`

3. Once you have added your files, you need to commit your changes. Always **create a very meaningful commit message related to the changes that you have done. Try to write the commit message in present imperative tense. Also namespace the commit message so that it becomes self-explanatory by just looking at the commit message.** For example,

    Docs: Add verbose setup docs for ubuntu

Step 4: Creating a Pull Request

1. Before creating a Pull Request, you need to first rebase your branch with the upstream master.
2. To rebase your branch, use following commands: `git fetch upstream` `git rebase upstream/master`
3. After rebasing, push the changes to your forked repository. `git push origin branch_name`
4. After pushing the code, create a Pull Request.
5. When creating a pull request, be sure to add a comment including these keywords, and also mention any maintainer to reviewing it.

Note:

- If you have any doubts, don't hesitate to ask in the comments. You may also add in any relevant content.
- After the maintainers review your changes, fix the code as suggested. Don't forget to add, commit, and push your code to the same branch.

Once you have completed the above steps, you have successfully created a Pull Request to EvalAI.

4.2.3 Code Standards

4.2.4 Testing Guidelines

4.3 Deployment

4.3.1 Production Deployment

4.3.2 Worker Setup

4.3.3 Scaling Guidelines

4.4 Maintenance

4.4.1 Migrations

> Migrations are Django's way of propagating changes you make to your models (adding a field, deleting a model, etc.) into your database schema. They're designed to be mostly automatic, but you'll need to know when to make migrations, when to run them, and the common problems you might run into. - Django Migration Docs

Creating a migration

- We recommend you to create migrations per app, where the changes are only about a single issue or feature.

```
# migration only for jobs app
python manage.py makemigrations jobs
```

- Always create named migrations. You can name migrations by passing `-n` or `--name` argument

```
python manage.py makemigrations jobs -n=execution_time_limit
# or
python manage.py makemigrations jobs --name=execution_time_limit
```

- While creating migrations on local environment, don't forget to add development settings.

```
python manage.py makemigrations
```

The following is an example of a complete named migration for the `jobs` app, wherein a execution time limit field is added to the `Submission` model:

```
python manage.py makemigrations jobs --name=execution_time_limit
```

- Files create after running `makemigrations` should be committed along with other files
- While creating a migration for your concerned change, it may happen that some other changes are also there in the migration file. Like adding a `execution_time_limit` field on `Submission` model also brings in the change for `when_made_public` being added. In that case, open an new issue and clearly mention the issue over there. If possible fix the issue yourself, by opening a new branch and creating migrations only for the concerned part. The idea here is that a commit should only include its concerned migration changes and nothing else.

4.4.2 Monitoring

4.4.3 Backup Recovery

CHAPTER

FIVE

API REFERENCE

CHAPTER

SIX

EXAMPLES TUTORIALS

6.1 Host Examples

6.1.1 Simple Prediction

6.1.2 Code Upload RL

6.1.3 Multi Phase

6.1.4 Remote Evaluation

6.2 Participants Examples

6.2.1 First Submission

6.2.2 Team Collaboration

6.3 Integration Examples

6.3.1 Github Integration

6.3.2 Jupyter Notebook

6.3.3 Colab Integration

CHAPTER

SEVEN

TROUBLESHOOTING

7.1 Common Issues

7.1.1 Installation Issues

7.1.2 Submission Errors

7.1.3 Evaluation Issues

7.1.4 CLI Issues

7.2 Error Codes

7.2.1 Error Reference

7.2.2 Debugging Guide

7.3 Support

7.3.1 FAQs (Frequently Asked Questions)

This section provides answers to common questions developers have while working with or contributing to EvalAI. It is organized by category to help you find answers quickly.

Table of Contents

Setup & Installation

Q. How to start contributing?

EvalAI's issue tracker is good place to start. If you find something that interests you, comment on the thread and we'll help get you started. Alternatively, if you come across a new bug on the site, please file a new issue and comment if you would like to be assigned. Existing issues are tagged with one or more labels, based on the part of the website it touches, its importance etc., which can help you select one.

Q. What are the technologies that EvalAI uses?

Please refer to Technologies Used

Q. Why was `virtualenv` setup deprecated?

Due to evolving dependencies and environment complexity, we now recommend using Docker-based setup for reliability and consistency across systems.

Docker & Frontend Issues

Q. I see `Cannot GET \` on `http://localhost:8888/` when using Docker.

This may happen if the container is not built properly. Run:

```
docker compose down
docker compose up --build
```

Q. I get this error: `ERROR: Version in "./docker-compose.yml" is unsupported.`

Upgrade your Docker engine to the latest version compatible with Compose file version 3.

Q. Nothing happens after clicking Login on EvalAI dashboard

This usually happens due to cache. Clean your browser cache & cookies completely and try accessing the website again, if still doesn't work, then try on a new browser profile.

Q. While building EvalAI via Docker, I get:

```
ERROR: Service 'celery' failed to build: pull access denied for evalai_django,␣
↪repository does not exist or may require 'docker login': denied: requested access to␣
↪the resource is denied
```

Ensure that your directory is named `evalai` (all lowercase). Docker image naming depends on the folder name. For instance, the image evalai_django gets renamed to evalai_dev_django if your directory is renamed to EvalAI_dev.

Python & Backend Errors

Q. I get this error during DB seeding:

```
Exception while running run() in 'scripts.seed'
```

Try deleting the PostgreSQL database manually or ensure you're using Python 2.7.x (not Python 3.x).

Q. I see `Peer authentication failed for user "postgres"` when using `createdb`.

Try creating a new user and then grant all the privileges to it and then create a db.

Q. I get this error while running tests inside Docker:

```
import file mismatch...
```

Clean **pycache** and .pyc files:

```
find . | grep -E "(__pycache__|\.pyc|\.pyo$)" | xargs rm -rf
```

Development & Contribution

Q. How do I fix coverage decrease warnings?

This means your new code isn't covered by tests. Click on the coverage report to view uncovered lines and add test cases accordingly.

Logs & Debugging

Q. How do I debug `psycopg2` installation errors while using `pip install -r dev/requirement.txt`?

Error:

```
Error: You need to install postgresql-server-dev-X.Y...
```

Fix:

```
sudo apt-get install postgresql
sudo apt-get install python-psycopg2
sudo apt-get install libpq-dev
```

Common Node, npm & Frontend Issues

Q. `gulp: command not found`

```
npm install -g gulp-cli
```

Q. `bower: command not found`

```
npm install -g bower
```

Q. Mocha reporter not loading:

```
npm uninstall -g generator-karma
npm install -g generator-angular
```

Q. Error: `Gem sass is not installed` while executing `gulp dev:runserver`

```
gem install sass
```

Q. Getting `karma@>=0.9.0 but none was installed` while executing `npm install`:

Reinstall after removing cache:

```
npm uninstall karma
npm install karma
npm cache clean --force
```

Q. Getting:

```
/usr/lib//nodejs/gulp//bin/gulp.js:132
TypeError: Cannot read properly 'apply of undefined'
```

Fix:

```
rm -rf node_modules/ bower_components
npm install
bower install
```

Proxy / Network Issues

Q. npm install fails with ECONNRESET or tunneling error

Fix your proxy settings:

```
npm config delete proxy
npm config delete https-proxy
npm config set registry https://registry.npmjs.org/
```

PostgreSQL Errors

Q. ERROR: Port 5432 already in use

Check and kill the process:

```
sudo netstat -lpn | grep :5432
sudo kill <PID>
```

Learning Resources

- Git and GitHub Learning Resources
- Markdown Guide

Recommended Next Steps

- Refer to the EvalAI Docs
- Join our Slack to ask for help if you're stuck

7.3.2 Community Support

7.3.3 Contact Support

CHAPTER

EIGHT

REFERENCES

8.1 Glossary

8.1.1 Challenge

An event, run by an institute or organization, wherein a number of researchers, students, and data scientists participate and compete with each other over a period of time. Each challenge has a start time and generally an end time too.

8.1.2 Challenge host

A member of the host team who organizes a challenge. In our system, it is a form of representing a user. This user can be in the organizing team of many challenges, and hence for each challenge, its challenge host will be different.

8.1.3 Challenge host team

A group of challenge hosts who organizes a challenge. They are identified by a unique team name.

8.1.4 Challenge phase

A challenge phase represents a distinct stage of the challenge. Over a period of time, challenge organizers have started to use the challenge phase as a way to:

1. Decide when to evaluate submissions on a subset of the test-set or when to evaluate on the whole test-set (for e.g, VQA Challenge 2019)
2. Use different challenge phases as different tracks of the same challenge (for e.g., CARLA Autonomous Driving Challenge)

8.1.5 Challenge phase split

A challenge phase split is the relation between a challenge phase and dataset splits for a challenge with a many-to-many relation. This is used to set the privacy of submissions (public/private) to different dataset splits for different challenge phases.

8.1.6 Dataset

A dataset in EvalAI is the main entity in which an AI challenge is based on. Participants are expected to make submissions corresponding to different splits of the corresponding dataset.

8.1.7 Dataset split

A dataset is generally divided into different parts called dataset split. Generally, a dataset has three different splits:

- Training set
- Validation set
- Test set

8.1.8 EvalAI

EvalAI is an open-source web platform that aims to be the state of the art in AI. Its goal is to help AI researchers, practitioners, and students to host, collaborate, and participate in AI challenges organized around the globe.

8.1.9 Leaderboard

The leaderboard can be defined as a scoreboard listing the names of the teams along with their current scores. Currently, each challenge has its own leaderboard.

8.1.10 Phase

A challenge can be divided into many phases (or challenge phases). A challenge phase can have the same or different start and end date than the challenge start and end date.

8.1.11 Participant

A member of the team competing against other teams for any particular challenge. It is a form of representing a user. A user can participate in many challenges, hence for each challenge, its participant entry will be different.

8.1.12 Participant team

A group of one or more participants who are taking part in a challenge. They are identified uniquely by a team name.

8.1.13 Submission

A way of submitting your results to the platform, so that it can be evaluated and ranked amongst others. A submission can be public or private, depending on the challenge.

8.1.14 Submission worker

A python script which processes submission messages received from a queue. It does the heavy lifting task of receiving a submission, performing mandatory checks, and then evaluating the submission and updating its status in the database.

8.1.15 Team

A model, present in `web` app, which helps CloudCV register new contributors as a core team member or simply an open source contributor.

8.1.16 Test annotation file

This is generally a file uploaded by a challenge host and is associated with a challenge phase. This file is used for ranking the submission made by a participant. An annotation file can be shared by more than one challenge phase. In the codebase, this is present as a file field attached to challenge phase model.

8.2 Cite

If you are using [EvalAI] for hosting challenges, please cite the following technical report:

```
@article{EvalAI,
    title   =  {EvalAI: Towards Better Evaluation Systems for AI Agents},
    author  =  {Deshraj Yadav and Rishabh Jain and Harsh Agrawal and Prithvijit
                Chattopadhyay and Taranjeet Singh and Akash Jain and Shiv Baran
                Singh and Stefan Lee and Dhruv Batra},
    year    =  {2019},
    volume  =  arXiv:1902.03570
}
```

8.3 Changelog

8.3.1 Deprecated Methods

Support for Docker image and code-upload based submissions is currently deprecated across both EvalAI and EvalAI CLI.

8.4 Roadmap

8.5 Configuration Reference

8.5.1 YAML Schema

8.5.2 Evaluation Metrics

8.5.3 Phase Settings

8.6 Legal

8.6.1 License

BSD License

For EvalAI software

Copyright (c) 2017-present, CloudCV.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
- Neither the name EvalAI nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS “AS IS” AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT

SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

8.6.2 Privacy Policy

8.6.3 Terms of Service

CHAPTER

NINE

INDICES AND TABLES

- genindex
- modindex
- search