Red Hat Enterprise Linux AI 1.4

Generating a custom LLM using RHEL AI

Using SDG, training, and evaluation to create a custom LLM

Last Updated: 2025-03-19

Red Hat Enterprise Linux AI 1.4 Generating a custom LLM using RHEL AI

Using SDG, training, and evaluation to create a custom LLM

Legal Notice

Copyright © 2025 Red Hat, Inc.

The text of and illustrations in this document are licensed by Red Hat under a Creative Commons Attribution–Share Alike 3.0 Unported license ("CC-BY-SA"). An explanation of CC-BY-SA is available at
http://creativecommons.org/licenses/by-sa/3.0/
. In accordance with CC-BY-SA, if you distribute this document or an adaptation of it, you must provide the URL for the original version.

Red Hat, as the licensor of this document, waives the right to enforce, and agrees not to assert, Section 4d of CC-BY-SA to the fullest extent permitted by applicable law.

Red Hat, Red Hat Enterprise Linux, the Shadowman logo, the Red Hat logo, JBoss, OpenShift, Fedora, the Infinity logo, and RHCE are trademarks of Red Hat, Inc., registered in the United States and other countries.

Linux ® is the registered trademark of Linus Torvalds in the United States and other countries.

Java ® is a registered trademark of Oracle and/or its affiliates.

XFS ® is a trademark of Silicon Graphics International Corp. or its subsidiaries in the United States and/or other countries.

MySQL ® is a registered trademark of MySQL AB in the United States, the European Union and other countries.

Node.js ® is an official trademark of Joyent. Red Hat is not formally related to or endorsed by the official Joyent Node.js open source or commercial project.

The OpenStack ® Word Mark and OpenStack logo are either registered trademarks/service marks or trademarks/service marks of the OpenStack Foundation, in the United States and other countries and are used with the OpenStack Foundation's permission. We are not affiliated with, endorsed or sponsored by the OpenStack Foundation, or the OpenStack community.

All other trademarks are the property of their respective owners.

Abstract

This document provides instructions on how users can generate a custom LLM through SDG, training and evaluation

Table of Contents

CHAPTER 1. GENERATING A NEW DATASET WITH SDG

After customizing your taxonomy tree, you can generate a synthetic dataset using the Synthetic Data Generation (SDG) process on Red Hat Enterprise Linux AI. SDG is a process that creates an artificially generated dataset that mimics real data based on provided examples. SDG uses a YAML file containing question-and-answer pairs as input data. With these examples, SDG utilizes the **mixtral-8x7b-instruct-v0-1** LLM as a teacher model to generate similar question-and-answer pairs. In the SDG pipeline, many questions are generated and scored based on quality, where the **mixtral-8x7b-instruct-v0-1** teacher model assesses their relevance and coherence. The pipeline then applies a filtering mechanism to select the highest-scoring questions, generates corresponding answers, and further evaluates their accuracy based on the original example question. The final set of high-quality question-and-answer pairs is then included in the synthetic dataset used for training.

1.1. CREATING A SYNTHETIC DATASET USING YOUR EXAMPLES

You can use your examples and run the SDG process to create a synthetic dataset.

IMPORTANT

If you are running SDG on a system with 4xL40s, you must use the following parameters for SDG to run properly.

```
ilab data generate --num-cpus 4
```

Prerequisites

- You installed RHEL AI with the bootable container image.
- You created a custom **qna.yaml** file with knowledge data.
- You downloaded the **mixtral-8x7b-instruct-v0-1** teacher model for SDG.
- You downloaded the **skills-adapter-v3:1.4** and **knowledge-adapter-v3:1.4** LoRA layered skills and knowledge adapter.
- You have root user access on your machine.

Procedure

1. To generate a new synthetic dataset, based on your custom taxonomy with knowledge, run the following command:

   ```
   $ ilab data generate
   ```

   NOTE

   You can use the **--enable-serving-output** flag when running the **ilab data generate** command to display the vLLM startup logs.

   a. At the start of the SDG process, vLLM attempts to start a server for hosting the **mixtral-8x7B-instruct** teacher model.

   Example output of vLLM attempting to start a server

```
Starting a temporary vLLM server at http://127.0.0.1:47825/v1
INFO 2024-08-22 17:01:09,461 instructlab.model.backends.backends:480: Waiting for
the vLLM server to start at http://127.0.0.1:47825/v1, this might take a moment...
Attempt: 1/120
INFO 2024-08-22 17:01:14,213 instructlab.model.backends.backends:480: Waiting for
the vLLM server to start at http://127.0.0.1:47825/v1, this might take a moment...
Attempt: 2/120
```

b. Once vLLM connects, the SDG process starts creating synthetic data based on your seed examples in the **qna.yaml** file.

Example output of vLLM connecting and SDG generating

```
INFO 2024-08-22 15:16:43,497 instructlab.model.backends.backends:480: Waiting for
the vLLM server to start at http://127.0.0.1:49311/v1, this might take a moment...
Attempt: 74/120
INFO 2024-08-22 15:16:45,949 instructlab.model.backends.backends:487: vLLM engine
successfully started at http://127.0.0.1:49311/v1
Generating synthetic data using '/usr/share/instructlab/sdg/pipelines/agentic' pipeline,
'/var/home/cloud-user/.cache/instructlab/models/mixtral-8x7b-instruct-v0-1' model,
'/var/home/cloud-user/.local/share/instructlab/taxonomy' taxonomy, against
http://127.0.0.1:49311/v1 server
INFO 2024-08-22 15:16:46,594 instructlab.sdg:375: Synthesizing new instructions. If you
aren't satisfied with the generated instructions, interrupt training (Ctrl-C) and try adjusting
your YAML files. Adding more examples may help.
```

2. The SDG process completes when the CLI displays the location of your new data set.

Example output of a successful SDG run

```
INFO 2024-08-16 17:12:46,548 instructlab.sdg.datamixing:200: Mixed Dataset saved to
/home/example-user/.local/share/instructlab/datasets/skills_train_msgs_2024-08-
16T16_50_11.jsonl
INFO 2024-08-16 17:12:46,549 instructlab.sdg:438: Generation took 1355.74s
```

NOTE

This process can be time consuming depending on your hardware specifications.

Verification

1. To verify that the SDG files were created, navigate to your **~/.local/share/instructlab/datasets/** directory and list the files corresponding to the date when the data was generated. For example:

```
$ ls 2024-03-24_194933
```

Example output

```
knowledge_recipe_2024-03-24T20_54_21.yaml                skills_recipe_2024-03-
24T20_54_21.yaml
knowledge_train_msgs_2024-03-24T20_54_21.jsonl            skills_train_msgs_2024-03-
```

```
24T20_54_21.jsonl
messages_granite-7b-lab-Q4_K_M_2024-03-24T20_54_21.jsonl    node_datasets_2024-03-
24T15_12_12/
```

IMPORTANT

Make a note of your most recent **knowledge_train_msgs.jsonl** and **skills_train_msgs.jsonl** file. You need to specify this file during multi-phase training. Each JSONL has the time stamp on the file, for example **knowledge_train_msgs_2024-08-08T20_04_28.jsonl**, use the most recent file when training.

2. Optional: You can view output of SDG by navigating to the **~/.local/share/datasets/<generation-date>** directory and opening the **JSONL** file.

```
$ cat ~/.local/share/datasets/<generation-date>/<jsonl-dataset>
```

Example output of a SDG JSONL file

```
{"messages":[{"content":"I am, Red Hat\u00ae Instruct Model based on Granite 7B, an AI
language model developed by Red Hat and IBM Research, based on the Granite-7b-base
language model. My primary function is to be a chat assistant.","role":"system"},{"content":"
<|user|>\n### Deep-sky objects\n\nThe constellation does not lie on the [galactic\nplane]
(galactic_plane \"wikilink\") of the Milky Way, and there are no\nprominent star clusters. [NGC
625](NGC_625 \"wikilink\") is a dwarf\n[irregular galaxy](irregular_galaxy \"wikilink\") of
apparent magnitude\n11.0 and lying some 12.7 million light years distant.
Only 24000 light\nyears in diameter, it is an outlying member of the [Sculptor\nGroup]
(Sculptor_Group \"wikilink\"). NGC 625 is thought to have been\ninvolved in a collision and is
experiencing a burst of [active star\nformation](Active_galactic_nucleus \"wikilink\").
[NGC\n37](NGC_37 \"wikilink\") is a [lenticular\ngalaxy](lenticular_galaxy \"wikilink\") of
apparent magnitude 14.66. It is\napproximately 42 [kiloparsecs](kiloparsecs \"wikilink\")
(137,000\n[light-years](light-years \"wikilink\")) in diameter and about 12.9\nbillion years old.
[Robert's Quartet](Robert's_Quartet \"wikilink\")\n(composed of the
irregular galaxy [NGC 87](NGC_87 \"wikilink\"), and three\nspiral galaxies [NGC 88](NGC_88
\"wikilink\"), [NGC 89](NGC_89 \"wikilink\")\nand [NGC 92](NGC_92 \"wikilink\")) is a group of
four galaxies located\naround 160 million light-years away which are in the process
of\ncolliding and merging. They are within a circle of radius of 1.6 arcmin,\ncorresponding to
about 75,000 light-years. Located in the galaxy ESO\n243-49 is [HLX-1](HLX-1 \"wikilink\"),
an [intermediate-mass black\nhole](intermediate-mass_black_hole \"wikilink\")\u2014the first
one of its kind\nidentified. It is thought to be a remnant of a dwarf
galaxy that was\nabsorbed in a [collision](Interacting_galaxy \"wikilink\") with ESO\n243-49.
Before its discovery, this class of black hole was only\nhypothesized.\n\nLying within the
bounds of the constellation is the gigantic [Phoenix\ncluster](Phoenix_cluster \"wikilink\"),
which is around 7.3 million light\nyears wide and 5.7 billion light years away, making it one of
the most\nmassive [galaxy clusters](galaxy_cluster \"wikilink\"). It was first\ndiscovered in
2010, and the central galaxy is producing an estimated 740\nnew stars a year. Larger still is
[El\nGordo](El_Gordo_(galaxy_cluster) \"wikilink\"),
or officially ACT-CL\nJ0102-4915, whose discovery was announced in 2012.
```

1.2. RUNNING SYNTHETIC DATA GENERATION (SDG) IN THE BACKGROUND

There are various ways you can manage and interact with the SDG process.

Running SDG in the background allows you to continue using your terminal while SDG is still running.

Prerequisites

- You installed RHEL AI with the bootable container image.
- You created a custom **qna.yaml** file with knowledge data.
- You downloaded the **mixtral-8x7b-instruct-v0-1** teacher model for SDG.
- You downloaded the **skills-adapter-v3:1.4** and **knowledge-adapter-v3:1.4** LoRA layered skills and knowledge adapter.
- You have root user access on your machine.

Procedure

1. To start an SDG process in the background, run the following command:

   ```
   $ ilab data generate -dt
   ```

   #### Example output of a successful start

   ```
   $ INFO 2025-01-15 11:36:47,557 instructlab.process.process:236: Started subprocess with PID 68289. Logs are being written to /Users/<user-name>/.local/share/instructlab/logs/generation/generation-e85623ac-d35e-11ef-bc70-2a1c6126d703.log.
   ```

2. There are a few ways you can manage, view, and interact with a detached SDG process.

   - You can view all the current running processes and their status by entering the following command:

     ```
     $ ilab process list
     ```

     #### Example output of the listed processes

     ```
     +------------+-------+--------------------------------------+------------------------------------------------------------------------------------------------------------+----------+---------+
     | Type       | PID   | UUID                                 | Log File                                                                                                   | Runtime  | Status  |
     +------------+-------+--------------------------------------+------------------------------------------------------------------------------------------------------------+----------+---------+
     | Generation | 30334 | f2623406-de55-11ef-b684-2a1c6126d703 | /Users/<user-name>/.local/share/instructlab/logs/generation/generation-f2623406-de55-11ef-b684-2a1c6126d703.log| 00:08:30 | Running |
     +------------+-------+--------------------------------------+------------------------------------------------------------------------------------------------------------+----------+---------+
     ```

   - You can join and view the latest process by running the following command:

WARNING

You cannot detach from an SDG process once already attached.

```
$ ilab process attach --latest
```

CHAPTER 2. TRAINING A MODEL

RHEL AI can use your taxonomy tree and synthetic data to create a newly trained model with your domain-specific knowledge or skills using multi-phase training and evaluation. You can run the full training and evaluation process using the synthetic dataset you generated. The LAB optimized technique of multi-phase training is a type of LLM training that goes through multiple stages of training and evaluation. In these various stages, RHEL AI runs the training process and produces model checkpoints. The best checkpoint is selected for the next phase. This process creates many checkpoints and selects the best scored checkpoint. This best scored checkpoint is your newly trained LLM.

The entire process creates a newly generated model that is trained and evaluated using the synthetic data from your taxonomy tree.

2.1. TRAINING THE MODEL ON YOUR DATA

You can use Red Hat Enterprise Linux AI to train a model with your synthetically generated data. The following procedures show how to do this using the LAB multi-phase training strategy.

IMPORTANT

Red Hat Enterprise Linux AI general availability does not support training and inference serving at the same time. If you have an inference server running, you must close it before you start the training process.

Prerequisites

- You installed RHEL AI with the bootable container image.
- You downloaded the **granite-7b-starter** model.
- You created a custom **qna.yaml** file with knowledge data.
- You ran the synthetic data generation (SDG) process.
- You downloaded the **prometheus-8x7b-v2-0** judge model.
- You have root user access on your machine.

Procedure

1. You can run multi-phase training and evaluation by running the following command with the data files generated from SDG.

   NOTE

   You can use the **--enable-serving-output** flag with the **ilab model train** commmand to display the training logs.

```
$ ilab model train --strategy lab-multiphase \
        --phased-phase1-data ~/.local/share/instructlab/datasets/<generation-date>/<knowledge-train-messages-jsonl-file> \
        --phased-phase2-data ~/.local/share/instructlab/datasets/<generation-date>/<skills-train-messages-jsonl-file>
```

where

<generation-date>
: The date of when you ran Synthetic Data Generation (SDG).

<knowledge-train-messages-file>
: The location of the **knowledge_messages.jsonl** file generated during SDG. RHEL AI trains the student model **granite-7b-starter** using the data from this **.jsonl** file. Example path: **~/.local/share/instructlab/datasets/2024-09-07_194933/knowledge_train_msgs_2024-09-07T20_54_21.jsonl**.

<skills-train-messages-file>
: The location of the **skills_messages.jsonl** file generated during SDG. RHEL AI trains the student model **granite-7b-starter** using the data from the **.jsonl** file. Example path: **~/.local/share/instructlab/datasets/2024-09-07_194933/skills_train_msgs_2024-09-07T20_54_21.jsonl**.

NOTE

You can use the **--strategy lab-skills-only** value to train a model on skills only.

Example skills only training command:

```
$ ilab model train --strategy lab-skills-only --phased-phase2-data
~/.local/share/instructlab/datasets/<skills-train-messages-jsonl-file>
```

a. The first phase trains the model using the synthetic data from your knowledge contribution.

Example output of training knowledge

```
Training Phase 1/2...
TrainingArgs for current phase: TrainingArgs(model_path='/opt/app-
root/src/.cache/instructlab/models/granite-7b-starter', chat_tmpl_path='/opt/app-
root/lib64/python3.11/site-packages/instructlab/training/chat_templates/ibm_generic_tmpl.py',
data_path='/tmp/jul19-knowledge-26k.jsonl', ckpt_output_dir='/tmp/e2e/phase1/checkpoints',
data_output_dir='/opt/app-root/src/.local/share/instructlab/internal', max_seq_len=4096,
max_batch_len=55000, num_epochs=2, effective_batch_size=128, save_samples=0,
learning_rate=2e-05, warmup_steps=25, is_padding_free=True, random_seed=42,
checkpoint_at_epoch=True, mock_data=False, mock_data_len=0,
deepspeed_options=DeepSpeedOptions(cpu_offload_optimizer=False,
cpu_offload_optimizer_ratio=1.0, cpu_offload_optimizer_pin_memory=False,
save_samples=None), disable_flash_attn=False, lora=LoraOptions(rank=0, alpha=32,
dropout=0.1, target_modules=('q_proj', 'k_proj', 'v_proj', 'o_proj'), quantize_data_type=
<QuantizeDataType.NONE: None>))
```

b. Then, RHEL AI selects the best checkpoint to use for the next phase.

c. The next phase trains the model using the synthetic data from the skills data.

Example output of training skills

```
Training Phase 2/2...
TrainingArgs for current phase:
TrainingArgs(model_path='/tmp/e2e/phase1/checkpoints/hf_format/samples_52096',
chat_tmpl_path='/opt/app-root/lib64/python3.11/site-
packages/instructlab/training/chat_templates/ibm_generic_tmpl.py',
data_path='/usr/share/instructlab/sdg/datasets/skills.jsonl',
ckpt_output_dir='/tmp/e2e/phase2/checkpoints', data_output_dir='/opt/app-
root/src/.local/share/instructlab/internal', max_seq_len=4096, max_batch_len=55000,
num_epochs=2, effective_batch_size=3840, save_samples=0, learning_rate=2e-05,
warmup_steps=25, is_padding_free=True, random_seed=42, checkpoint_at_epoch=True,
mock_data=False, mock_data_len=0,
deepspeed_options=DeepSpeedOptions(cpu_offload_optimizer=False,
cpu_offload_optimizer_ratio=1.0, cpu_offload_optimizer_pin_memory=False,
save_samples=None), disable_flash_attn=False, lora=LoraOptions(rank=0, alpha=32,
dropout=0.1, target_modules=('q_proj', 'k_proj', 'v_proj', 'o_proj'), quantize_data_type=
<QuantizeDataType.NONE: None>))
```

d. Then, RHEL AI evaluates all of the checkpoints from phase 2 model training using the Multi-turn Benchmark (MT-Bench) and returns the best performing checkpoint as the fully trained output model.

Example output of evaluating skills

```
MT-Bench evaluation for Phase 2...
Using gpus from --gpus or evaluate config and ignoring --tensor-parallel-size configured in
serve vllm_args
INFO 2024-08-15 10:04:51,065 instructlab.model.backends.backends:437: Trying to connect
to model server at http://127.0.0.1:8000/v1
INFO 2024-08-15 10:04:53,580 instructlab.model.backends.vllm:208: vLLM starting up on pid
79388 at http://127.0.0.1:54265/v1
INFO 2024-08-15 10:04:53,580 instructlab.model.backends.backends:450: Starting a
temporary vLLM server at http://127.0.0.1:54265/v1
INFO 2024-08-15 10:04:53,580 instructlab.model.backends.backends:465: Waiting for the
vLLM server to start at http://127.0.0.1:54265/v1, this might take a moment... Attempt: 1/300
INFO 2024-08-15 10:04:58,003 instructlab.model.backends.backends:465: Waiting for the
vLLM server to start at http://127.0.0.1:54265/v1, this might take a moment... Attempt: 2/300
INFO 2024-08-15 10:05:02,314 instructlab.model.backends.backends:465: Waiting for the
vLLM server to start at http://127.0.0.1:54265/v1, this might take a moment... Attempt: 3/300
moment... Attempt: 3/300
INFO 2024-08-15 10:06:07,611 instructlab.model.backends.backends:472: vLLM engine
successfully started at http://127.0.0.1:54265/v1
```

1. After training is complete, a confirmation appears and displays your best performed checkpoint.

Example output of a complete multi-phase training run

```
Training finished! Best final checkpoint: samples_1945 with score: 6.813759384
```

Make a note of this checkpoint because the path is necessary for evaluation and serving.

Verification

- When training a model with **ilab model train**, multiple checkpoints are saved with the **samples_** prefix based on how many data points they have been trained on. These are saved to the **~/.local/share/instructlab/phase/** directory.

```
$ ls ~/.local/share/instructlab/phase/<phase1-or-phase2>/checkpoints/
```

Example output of the new models

```
samples_1711 samples_1945 samples_1456 samples_1462 samples_1903
```

2.1.1. Continuing or restarting a training run

RHEL AI allows you to continue a training run that may have failed during multi-phase training. There are a few ways a training run can fail:

- The vLLM server may not start correctly.
- A accelerator or GPU may freeze, causing training to abort.
- There may be an error in your InstructLab **config.yaml** file.

When you run multi-phase training for the first time, the initial training data gets saved into a **journalfile.yaml** file. If necessary, this metadata in the file can be used to restart a failed training.

You can also restart a training run which clears the training data by following the CLI prompts when running multi-phase training.

Prerequisites

- You ran multi-phase training with your synthetic data and that failed.

Procedure

1. Run the multi-phase training command again.

```
$ ilab model train --strategy lab-multiphase \
        --phased-phase1-data ~/.local/share/instructlab/datasets/<generation-
date>/<knowledge-train-messages-jsonl-file> \
        --phased-phase2-data ~/.local/share/instructlab/datasets/<generation-date>/<skills-
train-messages-jsonl-file>
```

   The Red Hat Enterprise Linux AI CLI reads if the **journalfile.yaml** file exists and continues the training run from that point.

2. The CLI prompts you to continue for the previous training run, or start from the beginning.

   - Type **n** in your shell to continue from your previews training run.

```
Metadata (checkpoints, the training journal) may have been saved from a previous
training run.
By default, training will resume from this metadata if it exists
Alternatively, the metadata can be cleared, and training can start from scratch
Would you like to START TRAINING FROM THE BEGINNING? n
```

   - Type **y** into the terminal to restart a training run.

```
Metadata (checkpoints, the training journal) may have been saved from a previous
training run.
```

```
By default, training will resume from this metadata if it exists
Alternatively, the metadata can be cleared, and training can start from scratch
Would you like to START TRAINING FROM THE BEGINNING? y
```

Restarting also clears your systems cache of previous checkpoints, journal files and other training data.

CHAPTER 3. EVALUATING THE MODEL

If you want to measure the improvements of your new model, you can compare its performance to the base model with the evaluation process. You can also chat with the model directly to qualitatively identify whether the new model has learned the knowledge you created. If you want more quantitative results of the model improvements, you can run the evaluation process in the RHEL AI CLI.

3.1. EVALUATING YOUR NEW MODEL

You can run the evaluation process in the RHEL AI CLI with the following procedure.

Prerequisites

- You installed RHEL AI with the bootable container image.
- You created a custom **qna.yaml** file with skills or knowledge.
- You ran the synthetic data generation process.
- You trained the model using the RHEL AI training process.
- You downloaded the **prometheus-8x7b-v2-0** judge model.
- You have root user access on your machine.

Procedure

1. Navigate to your working Git branch where you created your **qna.yaml** file.
2. You can now run the evaluation process on different benchmarks. Each command needs the path to the trained **samples** model to evaluate, you can access these checkpoints in your **~/.local/share/instructlab/checkpoints** folder.
    a. **MMLU_BRANCH benchmark -** If you want to measure how your knowledge contributions have impacted your model, run the **mmlu_branch** benchmark by executing the following command:

```
$ ilab model evaluate --benchmark mmlu_branch
    --model ~/.local/share/instructlab/phased/phase2/checkpoints/hf_format/<checkpoint>
\
    --tasks-dir ~/.local/share/instructlab/datasets/<generation-date>/<node-dataset> \
    --base-model ~/.cache/instructlab/models/granite-7b-starter
```

where

<checkpoint>

Specify the best scored checkpoint file generated during multi-phase training

<node-dataset>

Specify the **node_datasets** directory that was generated during SDG, in the **~/.local/share/instructlab/datasets/<generation-date>** directory, with the same timestamps as the.jsonl files used for training the model.

Example output

```
# KNOWLEDGE EVALUATION REPORT

## BASE MODEL (SCORE)
/home/user/.cache/instructlab/models/instructlab/granite-7b-lab/ (0.74/1.0)

## MODEL (SCORE)
/home/user/local/share/instructlab/phased/phases2/checkpoints/hf_format/samples_665
0.78/1.0)

### IMPROVEMENTS (0.0 to 1.0):
1. tonsils: 0.74 -> 0.78 (+0.04)
```

b. **MT_BENCH_BRANCH benchmark -**If you want to measure how your skills contributions have impacted your model, run the **mt_bench_branch** benchmark by executing the following command:

```
$ ilab model evaluate \
    --benchmark mt_bench_branch \
    --model ~/.local/share/instructlab/phased/phase2/checkpoints/hf_format/<checkpoint>
\
    --judge-model ~/.cache/instructlab/models/prometheus-8x7b-v2-0 \
    --branch <worker-branch> \
    --base-branch <worker-branch>
```

where

<checkpoint>

Specify the best scored checkpoint file generated during multi-phase training.

<worker-branch>

Specify the branch you used when adding data to your taxonomy tree.

<num-gpus>

Specify the number of GPUs you want to use for evaluation.

Example output

```
# SKILL EVALUATION REPORT

## BASE MODEL (SCORE)
/home/user/.cache/instructlab/models/instructlab/granite-7b-lab (5.78/10.0)

## MODEL (SCORE)
/home/user/local/share/instructlab/phased/phases2/checkpoints/hf_format/samples_665
6.00/10.0)

### IMPROVEMENTS (0.0 to 10.0):
1. foundational_skills/reasoning/linguistics_reasoning/object_identification/qna.yaml:
4.0 -> 6.67 (+2.67)
2. foundational_skills/reasoning/theory_of_mind/qna.yaml: 3.12 -> 4.0 (+0.88)
3.
foundational_skills/reasoning/linguistics_reasoning/logical_sequence_of_words/qna.yam
: 9.33 -> 10.0 (+0.67)
4. foundational_skills/reasoning/logical_reasoning/tabular/qna.yaml: 5.67 -> 6.33
```

```
(+0.67)
5. foundational_skills/reasoning/common_sense_reasoning/qna.yaml: 1.67 -> 2.33
(+0.67)
6. foundational_skills/reasoning/logical_reasoning/causal/qna.yaml: 5.67 -> 6.0
(+0.33)
7. foundational_skills/reasoning/logical_reasoning/general/qna.yaml: 6.6 -> 6.8 (+0.2)
8. compositional_skills/writing/grounded/editing/content/qna.yaml: 6.8 -> 7.0 (+0.2)
9. compositional_skills/general/synonyms/qna.yaml: 4.5 -> 4.67 (+0.17)

### REGRESSIONS (0.0 to 10.0):
1.
foundational_skills/reasoning/unconventional_reasoning/lower_score_wins/qna.yaml:
5.67 -> 4.0 (-1.67)
2. foundational_skills/reasoning/mathematical_reasoning/qna.yaml: 7.33 -> 6.0 (-1.33)
3. foundational_skills/reasoning/temporal_reasoning/qna.yaml: 5.67 -> 4.67 (-1.0)

### NO CHANGE (0.0 to 10.0):
1. foundational_skills/reasoning/linguistics_reasoning/odd_one_out/qna.yaml (9.33)
2. compositional_skills/grounded/linguistics/inclusion/qna.yaml (6.5)
```

3. Optional: You can manually evaluate each checkpoint using the MMLU and MT_BENCH benchmarks. You can evaluate any model against the standardized set of knowledge or skills, allowing you to compare the scores of your own model against other LLMs.

   a. **MMLU -** If you want to see the evaluation score of your new model against a standardized set of knowledge data, set the **mmlu** benchmark by running the following command:

   ```
   $ ilab model evaluate --benchmark mmlu --model
   ~/.local/share/instructlab/phased/phase2/checkpoints/hf_format/samples_665
   ```

   where

   <checkpoint>

   Specify one of the checkpoint files generated during multi-phase training.

   Example output

   ```
   # KNOWLEDGE EVALUATION REPORT

   ## MODEL (SCORE)
   /home/user/.local/share/instructlab/phased/phase2/checkpoints/hf_format/samples_665


   ### SCORES (0.0 to 1.0):
   mmlu_abstract_algebra - 0.31
   mmlu_anatomy - 0.46
   mmlu_astronomy - 0.52
   mmlu_business_ethics - 0.55
   mmlu_clinical_knowledge - 0.57
   mmlu_college_biology - 0.56
   mmlu_college_chemistry - 0.38
   mmlu_college_computer_science - 0.46
   ...
   ```

b. **MT_BENCH -** If you want to see the evaluation score of your new model against a standardized set of skills, set the **mt_bench** benchmark by running the following command:

```
$ ilab model evaluate --benchmark mt_bench --model
~/.local/share/instructlab/phased/phases2/checkpoints/hf_format/samples_665
```

where

<checkpoint>

Specify one of the checkpoint files generated during multi-phase training.

Example output

```
# SKILL EVALUATION REPORT

## MODEL (SCORE)
/home/user/local/share/instructlab/phased/phases2/checkpoints/hf_format/samples_665
7.27/10.0)

### TURN ONE (0.0 to 10.0):
7.48

### TURN TWO (0.0 to 10.0):
7.05
```

3.1.1. Domain-Knowledge benchmark evaluation

IMPORTANT

Domain-Knowledge benchmark evaluation is a Technology Preview feature only. Technology Preview features are not supported with Red Hat production service level agreements (SLAs) and might not be functionally complete. Red Hat does not recommend using them in production. These features provide early access to upcoming product features, enabling customers to test functionality and provide feedback during the development process.

For more information about the support scope of Red Hat Technology Preview features, see Technology Preview Features Support Scope .

The current knowledge evaluation benchmark in RHEL AI, MMLU and MMLU_branch, evaluates models on their ability to answer multiple choice questions. There was no way to give the model credit on moderately correct or incorrect answers.

The Domain-Knowledge benchmark (DK-bench) evaluation provides the ability to bring custom evaluation questions and score the models answers on a scale.

Each response given is compared to the reference answer and graded on the following scale by the judge model:

Table 3.1. Domain-Knowledge benchmark rubric

Score	Criteria
1	The response is entirely incorrect, irrelevant, or does not align with the reference in any meaningful way.
2	The response partially matches the reference but contains major errors, significant omissions, or irrelevant information.
3	The response aligns with the reference overall but lacks sufficient detail, clarity, or contains minor inaccuracies.
4	The response is mostly accurate, aligns closely with the reference, and contains only minor issues or omissions.
5	The response is fully accurate, completely aligns with the reference, and is clear, thorough, and detailed.

Prerequisites

- You installed RHEL AI with the bootable container image.
- You trained the model using the RHEL AI training process.
- You have root user access on your machine.

Procedure

1. To utilize custom evaluation, you must create a **jsonl** file that includes every question you want to ask a model to answer and evaluate.

   #### Example DK-bench **jsonl** file

   ```
   {"user_input":"What is the capital of Canada?","reference":"The capital of Canada is Ottawa."}
   ```

   where

   user_input
   Contains the question for the model.

   reference
   Contains the answer to the question.

2. To run the DK-bench benchmark with your custom evaluation questions, run the following command:

   ```
   $ ilab model evaluate --benchmark dk_bench --input-questions <path-to-jsonl-file> --model <path-to-model>
   ```

   where

<path-to-jsonl-file>

Specify the path to your **jsonl** file that contains your questions and answers.

<path-to-model>

Specify the path to the model you want to evaluate.

Example command

```
$ ilab model evaluate --benchmark dk_bench --input-questions
/home/use/path/to/questions.jsonl --model ~/.cache/instructlab/models/instructlab/granite-
7b-lab
```

Example output of domain-Knowledge benchmark evaluation

```
# DK-BENCH REPORT

## MODEL: granite-7b-lab

Question #1:     5/5
Question #2:     5/5
Question #3:     5/5
Question #4:     5/5
Question #5:     2/5
Question #6:     3/5
Question #7:     2/5
Question #8:     3/5
Question #9:     5/5
Question #10:     5/5
----------------------------
Average Score:   4.00/5
Total Score:     40/50
```

CHAPTER 4. SERVING AND CHATTING WITH YOUR NEW MODEL

You must deploy the model to your machine by serving the model. This deploys the model and makes the model available for interacting and chatting.

4.1. SERVING THE NEW MODEL

To interact with your new model, you must activate the model in a machine through serving. The **ilab model serve** command starts a vLLM server that allows you to chat with the model.

Prerequisites

- You installed RHEL AI with the bootable container image.
- You initialized InstructLab.
- You customized your taxonomy tree, ran synthetic data generation, trained, and evaluated your new model.
- You need root user access on your machine.

Procedure

- You can serve the model by running the following command:

  ```
  $ ilab model serve --model-path <path-to-best-performed-checkpoint>
  ```

  where:

  <path-to-best-performed-checkpoint>

  Specify the full path to the checkpoint you built after training. Your new model is the best performed checkpoint with its file path displayed after training.

  #### Example command:

  ```
  $ ilab model serve --model-path
  ~/.local/share/instructlab/phased/phase2/checkpoints/hf_format/samples_1945/
  ```

  IMPORTANT

  Ensure you have a slash / at the end of your model path.

  #### Example output of the ilab model serve command

  ```
  $ ilab model serve --model-path
  ~/.local/share/instructlab/phased/phase2/checkpoints/hf_format/<checkpoint>
  INFO 2024-03-02 02:21:11,352 lab.py:201 Using model /home/example-
  user/.local/share/instructlab/checkpoints/hf_format/checkpoint_1945 with -1 gpu-layers
  and 4096 max context size.
  ```

```
Starting server process
After application startup complete see http://127.0.0.1:8000/docs for API.
Press CTRL+C to shut down the server.
```

4.2. CHATTING WITH THE NEW MODEL

You can chat with your model that has been trained with your data.

Prerequisites

- You installed RHEL AI with the bootable container image.
- You initialized InstructLab.
- You customized your taxonomy tree, ran synthetic data generated, trained and evaluated your new model.
- You served your checkpoint model.
- You need root user access on your machine.

Procedure

1. Since you are serving the model in one terminal window, you must open a new terminal window to chat with the model.
2. To chat with your new model, run the following command:

   ```
   $ ilab model chat --model <path-to-best-performed-checkpoint-file>
   ```

   where:

   <path-to-best-performed-checkpoint-file>

   Specify the new model checkpoint file you built after training. Your new model is the best performed checkpoint with its file path displayed after training.

   Example command:

   ```
   $ ilab model chat --model
   ~/.local/share/instructlab/phased/phase2/checkpoints/hf_format/samples_1945
   ```

3. Example output of the InstructLab chatbot

   ```
   $ ilab model chat
   ╭──────────────────────────────────────────────────────────────
   ──────────────────────────────────────────────────────────────
   system
   ──────────────────────────────────────────────────────────────
   ─────────────────────────────────────────────────────────────╮
   │ Welcome to InstructLab Chat w/ CHECKPOINT_1945 (type /h for help)
   │
   ╰──────────────────────────────────────────────────────────────
   ```

```
──────────────────────────────────────────────────────────────────────────────
──────────────────────────────────────────────────────────────────────────────
──────────────────────────────────────────────────────────────────────────────
───────┘
>>>
[S][default]
```

Type **exit** to leave the chatbot.