Partner Model Validation Guide

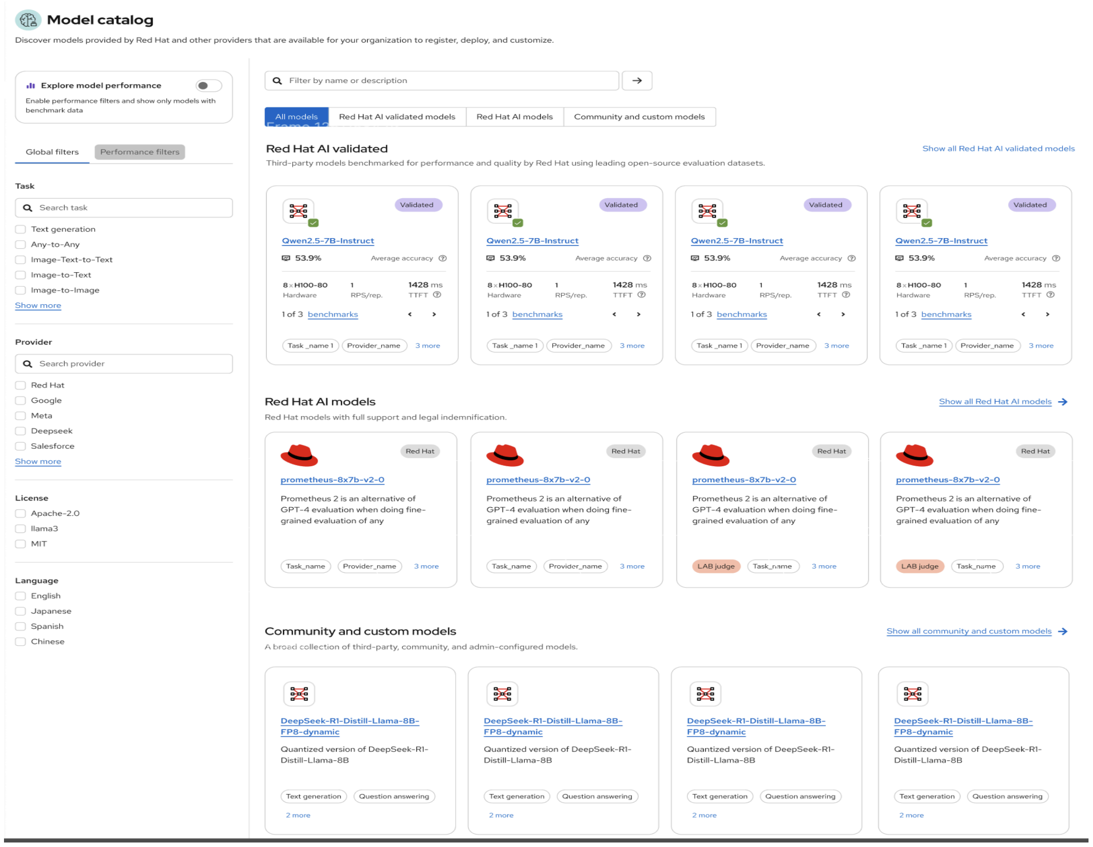
October 2025

Overview

Below is a step-by-step guide for Red Hat Partners who want to send over validation data for their own LLMs. Partner-validated LLMs will be surfaced in the Red Hat OpenShift AI Model Catalog in 3.0 and shipped to all Red Hat OpenShift AI customers in each product release with the model cards, performance benchmark data, and accuracy evaluations, providing Red Hat customers with model optionality and insights to leverage the best-in-class open source models. We outline the tooling that shall be used, as well as the schema of the data we expect to be included in the Red Hat OpenShift AI Model Catalog as a partner validated LLM.

If the vendor is not already a partner, then the vendor must become a Red Hat Partner (Build Module) before publication to the Red Hat OpenShift AI Model Catalog and/or Red Hat Ecosystem Catalog.

Model Catalog Screenshot:



Model catalog

Discover models provided by Red Hat and other providers that are available for your organization to register, deploy, and customize.

Explore model performance

Enable performance filters and show only models with benchmark data

Global filters | Performance filters

Task

Search task

- Text generation
- Any-to-Any
- Image-Text-to-Text
- Image-to-Text
- Image-to-Image

Show more

Provider

Search provider

- Red Hat
- Google
- Meta
- Deepseek
- Salesforce

Show more

License

- Apache-2.0
- llama3
- MIT

Language

- English
- Japanese
- Spanish
- Chinese

Filter by name or description

All models | Red Hat AI validated models | Red Hat AI models | Community and custom models

Red Hat AI validated

Third-party models benchmarked for performance and quality by Red Hat using leading open-source evaluation datasets.

Show all Red Hat AI validated models

Validated — Qwen2.5-7B-Instruct — 53.9% Average accuracy — 8 x H100-80 Hardware — 1 RPS/rep. — 1428 ms TTFT — 1 of 3 benchmarks — Task _name 1 — Provider_name — 3 more

Validated — Qwen2.5-7B-Instruct — 53.9% Average accuracy — 8 x H100-80 Hardware — 1 RPS/rep. — 1428 ms TTFT — 1 of 3 benchmarks — Task _name 1 — Provider_name — 3 more

Validated — Qwen2.5-7B-Instruct — 53.9% Average accuracy — 8 x H100-80 Hardware — 1 RPS/rep. — 1428 ms TTFT — 1 of 3 benchmarks — Task _name 1 — Provider_name — 3 more

Validated — Qwen2.5-7B-Instruct — 53.9% Average accuracy — 8 x H100-80 Hardware — 1 RPS/rep. — 1428 ms TTFT — 1 of 3 benchmarks — Task _name 1 — Provider_name — 3 more

Red Hat AI models

Red Hat models with full support and legal indemnification.

Show all Red Hat AI models

Red Hat — prometheus-8x7b-v2-0 — Prometheus 2 is an alternative of GPT-4 evaluation when doing fine-grained evaluation of any — Task_name — Provider_name — 3 more

Red Hat — prometheus-8x7b-v2-0 — Prometheus 2 is an alternative of GPT-4 evaluation when doing fine-grained evaluation of any — Task_name — Provider_name — 3 more

Red Hat — prometheus-8x7b-v2-0 — Prometheus 2 is an alternative of GPT-4 evaluation when doing fine-grained evaluation of any — LAB judge — Task_name — 3 more

Red Hat — prometheus-8x7b-v2-0 — Prometheus 2 is an alternative of GPT-4 evaluation when doing fine-grained evaluation of any — LAB judge — Task_name — 3 more

Community and custom models

A broad collection of third-party, community, and admin-configured models.

Show all community and custom models

DeepSeek-R1-Distill-Llama-8B-FP8-dynamic — Quantized version of DeepSeek-R1-Distill-Llama-8B — Text generation — Question answering — 2 more

DeepSeek-R1-Distill-Llama-8B-FP8-dynamic — Quantized version of DeepSeek-R1-Distill-Llama-8B — Text generation — Question answering — 2 more

DeepSeek-R1-Distill-Llama-8B-FP8-dynamic — Quantized version of DeepSeek-R1-Distill-Llama-8B — Text generation — Question answering — 2 more

DeepSeek-R1-Distill-Llama-8B-FP8-dynamic — Quantized version of DeepSeek-R1-Distill-Llama-8B — Text generation — Question answering — 2 more

Tooling

Our benchmarking pipeline relies on a specific set of open-source tools. Each tool serves a distinct purpose in the serving, performance testing, and evaluation phases.

vLLM

vLLM is an open-source, high-throughput, and memory-efficient inference and serving engine for LLMs. We use it to host the models and expose them through an OpenAI-compatible API endpoint. This standardization allows our benchmarking tools to interact with any model in a uniform way.

You should be running the latest vLLM image from Red Hat's quay.io found at:

https://catalog.redhat.com/en/software/containers/rhaiis/vllm-cuda-rhel9/6825c6b827e18fe3162148a9 - Cuda 3.2.2

https://catalog.redhat.com/en/software/containers/rhaiis/vllm-rocm-rhel9/6825c6b66db83eab748652de - ROCm 3.2.2

Upstream vLLM Version: v0.10.1.1

To serve your model using vLLM as an OpenAI-compatible server, run the following command. This will start a server for the specified model.

```
None

python3 -m vllm.entrypoints.openai.api_server \

  --model <your-model-name-or-path>
```

Replace <your-model-name-or-path> with the name of your model from the Hugging Face Hub or the local path to your model's weights.

GuideLLM

GuideLLM is an open-source performance benchmarking tool specifically designed to measure LLM inference-related metrics such as TTFT (Time To First Token), ITL (Inter Token Latency), E2E, and throughput.. It is used to understand the performance characteristics of the models under various load conditions.

The benchmarks should be run using the latest version of GuideLLM's image that can be found under *ghcr.io/vllm-project/guidellm*

Once your model is being served with vLLM, you can run a performance benchmark using GuideLLM. The following command runs a basic test to measure performance metrics.

To validate your model for specific use cases, please use the i/o token set up w/ GuideLLM outlined below.

The following scenarios are currently supported for partner-validation for the OpenShift AI Model Catalog:

- Chat (Input 512/ Output 256),
- Code fixing (Input 1024/ Output 1024),
- Long RAG (Input 10240/ Output 1536)
- RAG (Input 4096/ Output 512)

Each scenario will run with rps (requests per second) from 1 to 9, configured by the --rate flag in GuideLLM.

Run GuideLLM with the following command:

```shell
guidellm benchmark \
  --target http://localhost:8000 \
  --rate-type constant
  --rate $RPS
  --data $DATA_STRING
  --max-error-rate 0.5
  --max-seconds $MAX_SECONDS
  --output-dir ./benchmark_results
```

- --rate: The RPS (Request Per Second).
- --rate-type: Determines the GuideLLM mode of operation. constant means it's sending requests at a constant RPS.
- --max-seconds: Time the benchmark runs. Should be at least 5 minutes (300) and preferably over 15 minutes (900).
- --max-error-rate: It is not an overall error rate but rather a sliding window, please do not adjust it.
- --data: Specifies the dataset to use.

```
None

samples = rps * max_seconds

data =
f"prompt_tokens={input_tokens},prompt_tokens_stdev={input_tokens*0.1},output_tokens={output_tokens},output_tokens_stdev={output_tokens*0.1},samples={samples}"
```

To validate your model for specific use cases, please use the i/o token set up w/ GuideLLM outlined above.

LM Evaluation Harness

The Language Model Evaluation Harness (lm-eval-harness) is a framework for running standardized evaluations on LLMs. It allows us to test models on a wide range of benchmarks covering reasoning, knowledge, and language understanding to assess the quality and accuracy of the model's responses.

Usage Command: To run evaluation benchmarks, you can use the lm-eval command and provide the following arguments:

- --model vllm: Specifies that we are using the vLLM framework for evaluation.
- --model_args: Provides arguments to configure the vLLM model loading, including these fields:
  - pretrained: model name/path.
  - tensor_parallel_size: number of gpus.
  - gpu_memory_utilization: fraction of total GPU memory reserved to model and KV-cache.
  - add_bos_token: whether to append the bos token to the input.
- --tasks: A comma-separated list of evaluation tasks to run.
- --num_fewshot: Number of examples (shots) to be added to the prompt.
- --apply_chat_template: Triggers the use of the model's chat template.
- --confirm_run_unsafe_code: Allows the execution of code generated by the model (used in evaluations of code generation).
- --output_path: The file where the evaluation results will be saved.

Notes:

- Most chat templates already append the bos token at the start of the prompt. So when using --apply_chat_template, one should not use add_bos_token.
- Some multiple choice tasks use loglikelihood calculations to estimate the model prediction. These calculations require more memory than standard text generation and this is not accounted for in vLLM's memory reservation. Hence, in these cases it is recommended to set gpu_memory_utilization lower (0.4 to 0.6) to avoid out-of-memory errors.

Evaluation Commands with `lm-eval-harness`:
OpenLLMv1 (includes MMLU, Hellaswag, Winogrande, ARC Challenge, TruthfulQA and GSM8k)

```
None
lm_eval \
 --model vllm \
 --model_args
"pretrained=<your-model-name-or-path>,tensor_parallel_size=<number-of-GPUs>,dty
pe=auto,gpu_memory_utilization=0.6,add_bos_token=True" \
 --tasks openllm \
 --batch_size auto \
 --output_path ./eval_results.json
```

OpenLLMv2 (includes BBH, GPQA, MuSR, Math-lvl5, MMLU-Pro and IFEval)

```
None
lm_eval \
 --model vllm \
 --model_args
"pretrained=<your-model-name-or-path>,tensor_parallel_size=<number-of-GPUs>,dty
pe=auto,gpu_memory_utilization=0.9" \
 --tasks leaderboard \
 --batch_size auto \
 --apply_chat_template \
 --output_path ./eval_results.json
```

HumanEval

```
None
lm_eval \
 --model vllm \
 --model_args
"pretrained=<your-model-name-or-path>,tensor_parallel_size=<number-of-GPUs>,dty
pe=auto,gpu_memory_utilization=0.9" \
 --tasks humaneval_64_instruct \
 --batch_size auto \
 --apply_chat_template \
 --confirm_run_unsafe_code \
 --output_path ./eval_results.json
```

Data Schema

Finalized Scheme

```
JSON
{
 "$schema": "http://json-schema.org/draft-07/schema#",
 "title": "Experiment",
 "type": "object",
 "properties": {
  "experiment_id": {
   "type": "string",
   "format": "uuid",
   "description": "Unique identifier of experiment (based on uuidv4 format)"
  },
  "experiment_type": {
   "type": "string",
   "description": "Type of test: performance OR evaluation"
  },
  "model": {
   "type": "string",
   "description": "The name that appears in HuggingFace e.g.
RedHatAI/Mistral-Small-3.1-24B-Instruct-2503-FP8-dynamic"
  },
  "inference_server": {
   "type": "string",
   "description": "s.a vLLM, TGI, etc"
```

```
    },
    "inference_server_version": {
      "type": "string",
              "description": "version used s.a rh-vllm-v1.1.2"
    },
    "container_image": {
     "type": "string",
     "description": "image used at the benchmark s.a vllm/vllm-openai"
    },
    "container_image_tag": {
     "type": "string",
             "description": "container's tag s.a v0.8.3"
    },
    "container_entrypoint": {
     "type": "string",
             "description": "the command executed in the container"
    },
    "inference_server_args": {
     "type": "object",
     "description": "The args used to run the inference server, as a json object. Args like
  --tensor-parallel-size=1, --max-model-len=16384. See YAMLs in
  https://github.com/neuralmagic/model-validation-configs/"
    },
    "accelerator_type": {
     "type": "string",
     "description": "The name of the GPU accelerator s.a L4, A100-40, H100, MI300x etc"
    },
    "accelerator_count": {
     "type": "integer",
     "description": "Number of GPUs on machine"
    },
    "accelerator_memory_gb": {
     "type": "integer",
     "description": "GBs of memory in hardware accelerator. For H100 for example, 80"
    },
    "machine_type": {
     "type": "string",
     "description": "Type of machine used s.a g4de"
    },
    "provider": {
      "type": "string",
      "description": "Where was the machine running, i.e IBM, Azure, on-prem, etc."
    },
    "report": {
     "type": "object",
     "description": "Output JSON for the test (performance or eval output json). Each
performance result will be the raw GuideLLM output JSON, containing a list of benchmark
results with varying RPS. The eval results will be the raw lm-evaluation-harness output JSON."
    },
    "timestamp": {
      "type": "string",
      "format": "date-time",
      "description": "DateTime of when the experiment was ran"
    }
  },
  "required": [
    "experiment_id",
    "experiment_type",
    "model",
    "inference_server",
    "inference_server_version",
    "container_image",
    "container_image_tag",
    "inference_server_args",
    "accelerator_type",
    "accelerator_count",
    "accelerator_memory_gb",
    "report",
    "timestamp"
  ]
```

```
}
```

Notes

- We use the same format for either performance or evaluation data, the *result* field is just a dump of the output JSON you receive from GuideLLM / lm-eval-harness

Monotonicity Checks

Partner's data will be validated by the Red Hat Model Validation team, and monotonicity checks will be run on it. Monotonicity checks are a sanity check mechanism used to assess the quality of the data. Specifically, it checks for monotonicity across different metrics.

For example, for a given model, we expect that a single H100 GPU will have lower TTFT metrics than a single L4 GPU. There are many such checks that we run based on historical data Red Hat has. The data that does not pass the monotonicity checks will not be ingested into Red Hat products.

Questions/Troubleshooting

For questions or troubleshooting issues with measuring benchmarks with GuideLLM or evaluations with LM-Eval-Harness, please contact @rgreenberg@redhat.com.

Glossary

vLLM: An open-source serving engine designed for high-throughput and memory-efficient LLM inference.
GuideLLM: An open-source performance benchmarking tool used to measure LLM inference metrics like throughput, TTFT, and ITL under various load conditions.
E2E (End-to-End) Latency: The total time taken from sending a request to the model to receiving the final token of the generated response. It measures the complete user experience time.

ITL (Inter-Token Latency): The average time delay between generating consecutive tokens in a response after the first token has been produced. It measures the streaming speed of the model.

Throughput (tokens/second): The rate at which the model can process and generate tokens, typically measured in tokens per second. It indicates the overall processing capacity of the inference server.

TTFT (Time To First Token): The time elapsed from when a request is sent to the model until the very first token of the response is received. It measures the initial responsiveness of the model.

LM Evaluation Harness: A standardized framework used to test and evaluate the accuracy, reasoning, and knowledge of large language models across a wide range of academic benchmarks.

Model Card: A .md file that provides key information about an AI model, such as performance metrics, accuracy evaluations, intended use cases, and ethical considerations.

Monotonicity Checks: A data validation process to ensure that performance metrics scale logically and predictably with changes in hardware or configuration. For example, a more powerful GPU is expected to yield better performance (e.g., lower latency) than a less powerful one. Data that fails these "common sense" checks is considered unreliable.

RAG (Retrieval-Augmented Generation): A technique where an LLM's knowledge is supplemented by retrieving relevant information from an external knowledge base before generating a response. This allows the model to provide more accurate, up-to-date, and context-specific answers.

RPS (Requests Per Second): A measure of the load on a system, indicating how many requests are sent to the model every second. It's a key parameter for performance testing.