import mii

mii.deploy(task='text-generation', model="meta-llama/Llama-3.2-1B", deployment\_name="llama32", enable\_deepspeed=True)

import deepspeed

model = AutoModelForCausalLM.from\_pretrained(..., torch\_dtype=torch.float16).half()

ds\_model = deepspeed.init\_inference(model, dtype=torch.float16, mp\_size=1) # initialize optimized model

# Warmup

input\_ids = tokenizer("Hello world", return\_tensors='pt').input\_ids.cuda()

\_ = ds\_model(input\_ids)

torch.cuda.synchronize()

start = time.time()

\_ = ds\_model(input\_ids)

torch.cuda.synchronize()

print("Latency:", time.time() - start)

import requests, time

prompt = "Once upon a time,"

t0 = time.time()

response = requests.post("http://localhost:80/generate", json={"inputs": prompt, "max\_new\_tokens": 100})

dt = time.time() - t0

tokens\_generated = len(response.json()[0]['generated\_text'].split())

print(f"Generated {tokens\_generated} tokens in {dt:.2f}s, throughput {tokens\_generated/dt:.2f} tokens/sec")

Got it! I’ll gather a list of libraries and tools that you can use to benchmark HuggingFace-loaded models like LLaMA 3.2 and DeepSeek R1, especially for tasks like perplexity evaluation, standard NLP benchmarks (e.g., GLUE, SQuAD), and other LM benchmarks. I’ll also include usage examples and considerations for each tool.

I’ll get back to you shortly with a full overview.

**Benchmarking Libraries for Causal Language Models**

Evaluating causal language models (like LLaMA 3.2 or DeepSeek R1) requires tools that cover a range of metrics and tasks – from basic perplexity to complex benchmarks (GLUE, SQuAD, MMLU, etc.), as well as speed and memory usage. Below is a comprehensive guide to **benchmarking libraries and tools** for language models loaded via Hugging Face’s AutoModelForCausalLM. We organize them into **quality evaluation frameworks** (for accuracy and NLP tasks) and **performance benchmarking tools** (for speed/efficiency). Each tool includes what it benchmarks, supported models, Hugging Face integration, example usage, and pros/cons.

**Quality and Accuracy Evaluation Frameworks**

**Hugging Face Evaluate (Metrics & Perplexity)**

Hugging Face’s **Evaluate** library is a lightweight toolkit for computing evaluation metrics on models and datasets ( [Evaluate](https://huggingface.co/docs/evaluate/en/index#:~:text=A%20library%20for%20easily%20evaluating,machine%20learning%20models%20and%20datasets)). It provides dozens of metrics out-of-the-box for NLP tasks (and other domains) with a simple interface. For language models, you can use it to compute **perplexity** and task-specific scores (accuracy, F1, etc.) on standard benchmarks.

* **What it Benchmarks:** Individual metrics like perplexity, BLEU, accuracy, F1, SQuAD scores, etc. You can evaluate model **perplexity** on a corpus (a common LM metric) ([Perplexity - a Hugging Face Space by evaluate-metric](https://huggingface.co/spaces/evaluate-metric/perplexity#:~:text=Perplexity%20,likelihood%20of%20a)), or use it for **GLUE** and **SQuAD** by computing the appropriate metric on model predictions.
* **Supported Models:** Any Hugging Face model – you typically provide the model’s outputs or have the library load the model by ID to compute the metric.
* **Integration with Hugging Face:** *Evaluate* integrates seamlessly. For example, the built-in perplexity metric can directly load a model from the Hub by name. Usage might look like:
* import evaluate
* perplexity = evaluate.load("perplexity")
* result = perplexity.compute(model\_id="gpt2", data=["Hello world."])

This will load the gpt2 model and return its perplexity on the given text ([How can I use evaluate's perplexity metric on a model that's already loaded? - Intermediate - Hugging Face Forums](https://discuss.huggingface.co/t/how-can-i-use-evaluates-perplexity-metric-on-a-model-thats-already-loaded/48564#:~:text=perplexity%20%3D%20load%28,model)). For GLUE, you can load a GLUE metric (e.g. evaluate.load("glue", "sst2")) and compare your model’s predictions to labels.

* **Example Usage:** To get perplexity of a model on a text dataset:
* import evaluate
* ppl = evaluate.load("perplexity", module\_type="metric")
* result = ppl.compute(model\_id="EleutherAI/gpt-j-6B", data=my\_text\_list)
* print(result["perplexity"])

Similarly, for question answering (SQuAD) you can load evaluate.load("squad\_v2"), have the model answer questions, and compute exact match and F1.

* **Pros:** Very **easy to use** for computing standard metrics with one line ( [Evaluate](https://huggingface.co/docs/evaluate/en/index#:~:text=A%20library%20for%20easily%20evaluating,machine%20learning%20models%20and%20datasets)). Leverages Hugging Face’s extensive metric repository (including GLUE/SQuAD official metrics). Lightweight and flexible – you can plug in any model outputs. Good for **quick checks** like perplexity or accuracy on a given dataset.
* **Cons:** *Evaluate* by itself does **not automate the model inference** on a dataset (except for some metrics like perplexity where it can load the model for you). You often need to write the loop or use a pipeline to generate model predictions. It’s a metrics library, not a full benchmarking suite – so running a model on GLUE or MMLU requires additional code to handle data and prompting. In summary, it’s great for computing numbers, but not a one-click benchmarking solution for many tasks.

*Links:* Official docs ( [Evaluate](https://huggingface.co/docs/evaluate/en/index#:~:text=A%20library%20for%20easily%20evaluating,machine%20learning%20models%20and%20datasets)); Hugging Face hub for metrics (e.g., “perplexity” metric card).

**jiant (GLUE, SuperGLUE, and NLU Tasks)**

**jiant** is an NLP research toolkit (from NYU) designed for evaluating and fine-tuning models on a wide range of **natural language understanding (NLU) tasks** ([jiant is an NLP toolkit: Introducing jiant 2.0](https://wp.nyu.edu/cilvr/2020/10/07/jiant-is-an-nlp-toolkit-introducing-jiant-2-0/" \l ":~:text=,mll%2Fjiant)) ([jiant is an NLP toolkit: Introducing jiant 2.0](https://wp.nyu.edu/cilvr/2020/10/07/jiant-is-an-nlp-toolkit-introducing-jiant-2-0/" \l ":~:text=jiant%20supports%20more%20than%2050,a%20task%2C%20that%E2%80%99s%20easy%20too)). It was built for benchmarks like GLUE and SuperGLUE, making it highly relevant for standard tasks such as SST-2 sentiment, MNLI, QNLI, QQP, RTE, as well as question answering (SQuAD), etc.

* **What it Benchmarks:** Primarily **GLUE/SuperGLUE benchmarks** and related tasks (classification, QA, tagging, etc.). It supports 50+ NLU tasks including all of GLUE, SuperGLUE, and the XTREME multilingual tasks ([jiant is an NLP toolkit: Introducing jiant 2.0](https://wp.nyu.edu/cilvr/2020/10/07/jiant-is-an-nlp-toolkit-introducing-jiant-2-0/" \l ":~:text=,mll%2Fjiant)) ([jiant is an NLP toolkit: Introducing jiant 2.0](https://wp.nyu.edu/cilvr/2020/10/07/jiant-is-an-nlp-toolkit-introducing-jiant-2-0/" \l ":~:text=jiant%20supports%20more%20than%2050,a%20task%2C%20that%E2%80%99s%20easy%20too)). Tasks cover text classification, QA, entailment, coreference (WSC), etc.
* **Supported Models:** Originally designed for transformer encoders like BERT, RoBERTa, etc., via Hugging Face’s models ([jiant is an NLP toolkit: Introducing jiant 2.0](https://wp.nyu.edu/cilvr/2020/10/07/jiant-is-an-nlp-toolkit-introducing-jiant-2-0/" \l ":~:text=,mll%2Fjiant)) ([jiant is an NLP toolkit: Introducing jiant 2.0](https://wp.nyu.edu/cilvr/2020/10/07/jiant-is-an-nlp-toolkit-introducing-jiant-2-0/" \l ":~:text=Models)). jiant 2.0 integrates with Hugging Face Transformers, so it can use many **Hugging Face models** (it was built when decoder-only models were less common, but it does support models like BART as encoders). Newer models like LLaMA would need adaptation (LLaMA is a decoder-only LM, so to use it on GLUE one might need to add a classification head or treat it differently). Generally any model that can be an encoder for classification should work; out-of-the-box support exists for popular architectures (BERT, RoBERTa, ELECTRA, etc.) ([jiant is an NLP toolkit: Introducing jiant 2.0](https://wp.nyu.edu/cilvr/2020/10/07/jiant-is-an-nlp-toolkit-introducing-jiant-2-0/" \l ":~:text=,mll%2Fjiant)) ([jiant is an NLP toolkit: Introducing jiant 2.0](https://wp.nyu.edu/cilvr/2020/10/07/jiant-is-an-nlp-toolkit-introducing-jiant-2-0/" \l ":~:text=Models)).
* **Integration with Hugging Face:** jiant uses Hugging Face’s **datasets** and **transformers** under the hood. Tasks are pre-defined using HF datasets and it loads models via transformers APIs ([jiant is an NLP toolkit: Introducing jiant 2.0](https://wp.nyu.edu/cilvr/2020/10/07/jiant-is-an-nlp-toolkit-introducing-jiant-2-0/" \l ":~:text=and%20datasets%20%28formerly%20nlp%29)) ([jiant is an NLP toolkit: Introducing jiant 2.0](https://wp.nyu.edu/cilvr/2020/10/07/jiant-is-an-nlp-toolkit-introducing-jiant-2-0/" \l ":~:text=jiant%20supports%20more%20than%2050,a%20task%2C%20that%E2%80%99s%20easy%20too)). You typically configure a JSON/YAML to specify which pretrained model (by Hugging Face model name) to use for which task.
* **Example Usage:** After installing jiant, you might run a configuration to evaluate a model on RTE (a GLUE task). For example, using the CLI:
* python jiant/proj/main.py \
* --config\_path configs/roberta\_RTE.conf \
* --output\_path outputs/roberta\_RTE/

This would fine-tune and evaluate RoBERTa on the RTE dataset (for pure evaluation, you could use a pre-finetuned model). jiant also provides a quick-start that downloads a GLUE dataset and runs an eval. Another example: you can generate GLUE test predictions with --write\_test\_preds in the runscript ([GLUE Benchmark - nyu-mll/jiant · GitHub](https://github.com/nyu-mll/jiant/blob/master/guides/benchmarks/glue.md#:~:text=jiant%20supports%20generating%20submission%20files,py%20when%20running%20your%20workflow)).

* **Pros:** **Comprehensive NLU coverage** – ideal for standard benchmarks like GLUE. It handles all data preprocessing, model training, and evaluation in a unified way ([jiant is an NLP toolkit: Introducing jiant 2.0](https://wp.nyu.edu/cilvr/2020/10/07/jiant-is-an-nlp-toolkit-introducing-jiant-2-0/" \l ":~:text=Tasks)). **Multi-task capable**: you can evaluate or train on multiple tasks with one model. Good for research on transfer learning and multitask learning. Has a configuration-driven approach that ensures reproducibility.
* **Cons:** Focused on **classification and encoder-based models** – not specialized for causal LMs in few-shot settings. Using decoder-only LMs (like LLaMA) may require custom heads or prompting (which jiant wasn’t originally designed for). It’s a heavier framework to set up (designed for research workflows), which might be overkill if you just want to quickly get accuracy on one task. Also, jiant may not be actively updated for the very latest models or benchmarks (its peak was around the GLUE era, ~2020). For newer instruction-style tasks or generative benchmarks, other tools are more suitable.

*Links:* [jiant GitHub](https://github.com/nyu-mll/jiant) (with documentation and tutorials) ([jiant is an NLP toolkit: Introducing jiant 2.0](https://wp.nyu.edu/cilvr/2020/10/07/jiant-is-an-nlp-toolkit-introducing-jiant-2-0/" \l ":~:text=,mll%2Fjiant)).

**EleutherAI LM Evaluation Harness (Few-shot LM Benchmarks)**

The **EleutherAI LM Evaluation Harness** is a popular open-source framework for evaluating language models on a large number of benchmarks in a unified way ([GitHub - EleutherAI/lm-evaluation-harness: A framework for few-shot evaluation of language models.](https://github.com/EleutherAI/lm-evaluation-harness#:~:text=Overview)). It was originally developed to evaluate GPT-like models in a few-shot setting (without fine-tuning) on tasks ranging from question-answering to commonsense reasoning. This harness has been used widely in research and powers many LLM leaderboards.

* **What it Benchmarks:** Over **60 standard NLP benchmarks** (and hundreds of sub-tasks) for language models ([GitHub - EleutherAI/lm-evaluation-harness: A framework for few-shot evaluation of language models.](https://github.com/EleutherAI/lm-evaluation-harness#:~:text=Features%3A)). This includes **language-model-specific benchmarks** like *MMLU (Massive Multitask Language Understanding)*, **HellaSwag**, **ARC (Easy & Challenge)**, **TruthfulQA**, **PIQA**, **WinoGrande**, *BIG-Bench tasks*, and more ([LightEval Deep Dive: Hugging Face’s All-in-One Framework for LLM Evaluation - Cohorte Projects](https://www.cohorte.co/blog/lighteval-deep-dive-hugging-faces-all-in-one-framework-for-llm-evaluation" \l ":~:text=,60%2B%20benchmarks%20%28ARC)). It focuses on tasks that can be posed in a prompting/few-shot evaluation format. For example, it can evaluate multiple-choice QA (like ARC or MMLU) by prompting the model and checking accuracy, or open-ended generation tasks by comparing to references. (It does **not typically cover GLUE/SQuAD**, since those require fine-tuning or classification heads; the harness is more for zero-shot or few-shot *LM* evaluation.)
* **Supported Models:** It supports models from various sources – Hugging Face Transformers (any AutoModelForCausalLM essentially), as well as APIs (OpenAI GPT-4 via API), and optimized runtimes like vLLM ([GitHub - EleutherAI/lm-evaluation-harness: A framework for few-shot evaluation of language models.](https://github.com/EleutherAI/lm-evaluation-harness#:~:text=,available%20prompts%20ensures%20reproducibility%20and)). For Hugging Face models, it has a --model hf option (for a causal LM on the hub or local). It also supports large distributed models via DeepSpeed or GPT-NeoX libraries ([GitHub - EleutherAI/lm-evaluation-harness: A framework for few-shot evaluation of language models.](https://github.com/EleutherAI/lm-evaluation-harness#:~:text=,available%20prompts%20ensures%20reproducibility%20and)). Notably, it can handle models with **LoRA/PEFT adapters** and even quantized models (via GPTQ) ([GitHub - EleutherAI/lm-evaluation-harness: A framework for few-shot evaluation of language models.](https://github.com/EleutherAI/lm-evaluation-harness#:~:text=,LoRA%29%20supported%20in)). This means newer models like LLaMA 3.2 or DeepSeek R1 (if you have them as Hugging Face format) can be evaluated – though extremely large models like DeepSeek R1 (671B, MoE) might be impractical to run, the harness code itself can interface with them if the model is accessible.
* **Integration with Hugging Face:** Very close integration – if you specify --model hf and pretrained=<model\_name> it will use AutoModelForCausalLM.from\_pretrained(model\_name) internally ([lm-evaluation-harness - Codesandbox](http://codesandbox.io/p/github/jeremyw-dobeu/lm-evaluation-harness#:~:text=lm,batch_size%208)). It also uses the model’s tokenizer. Essentially, any model on the HF Hub or local path that AutoModelForCausalLM can load can be evaluated (the README explicitly notes support for HF transformers models and tokenization-agnostic handling) ([GitHub - EleutherAI/lm-evaluation-harness: A framework for few-shot evaluation of language models.](https://github.com/EleutherAI/lm-evaluation-harness#:~:text=,LoRA%29%20supported%20in)).
* **Example Usage:** You can use the harness via CLI or as a library. A typical CLI command is:
* pip install lm-eval
* lm\_eval --model hf \
* --model\_args pretrained=EleutherAI/gpt-j-6B,device=cuda:0 \
* --tasks hellaswag,arc\_challenge,mmlu \
* --num\_fewshot 5

This loads the GPT-J-6B model from HF and evaluates it on HellaSwag, ARC Challenge, and MMLU with 5-shot prompting (the few-shot examples are taken from each task’s prompt templates). The harness will output metrics like accuracy for each task. (You can also run it in zero-shot by --num\_fewshot 0.)

* **Pros:** **Wide coverage** – many of the standard academic *LLM benchmarks* are implemented and ready to use, which is ideal for evaluating general capabilities ([LightEval Deep Dive: Hugging Face’s All-in-One Framework for LLM Evaluation - Cohorte Projects](https://www.cohorte.co/blog/lighteval-deep-dive-hugging-faces-all-in-one-framework-for-llm-evaluation" \l ":~:text=,60%2B%20benchmarks%20%28ARC)) ([LightEval Deep Dive: Hugging Face’s All-in-One Framework for LLM Evaluation - Cohorte Projects](https://www.cohorte.co/blog/lighteval-deep-dive-hugging-faces-all-in-one-framework-for-llm-evaluation" \l ":~:text=EleutherAI%20LM%20Harness%20Python,framework%20with%20wide%20scenario%2Fmetric%20coverage)). It’s been **battle-tested** in the community (used for comparing GPT-3, PaLM, open models, etc.), ensuring reliable evaluation protocols. The harness is quite **extensible**: you can add new tasks or modify prompts easily, and it recently added features like config-driven prompts, batch evaluation, and even some multimodal support ([GitHub - EleutherAI/lm-evaluation-harness: A framework for few-shot evaluation of language models.](https://github.com/EleutherAI/lm-evaluation-harness#:~:text=,configurable%20fewshot%20settings%2C%20and%20more)) ([GitHub - EleutherAI/lm-evaluation-harness: A framework for few-shot evaluation of language models.](https://github.com/EleutherAI/lm-evaluation-harness#:~:text=,defined%20task%20groupings%2C%20and%20more)). It supports **Hugging Face models out-of-the-box**, and can even leverage faster backends like vLLM for speedups ([GitHub - EleutherAI/lm-evaluation-harness: A framework for few-shot evaluation of language models.](https://github.com/EleutherAI/lm-evaluation-harness#:~:text=,supported%20in%20HuggingFace%27s%20PEFT%20library)). Being a CLI tool, it’s straightforward to run and obtain a summary of results.
* **Cons:** The harness is a standalone tool, which means setting it up and learning its interface (task names, few-shot settings) has a slight learning curve. While it covers many tasks, it is **focused on *evaluation***; it won’t fine-tune your model, and tasks are generally approached in a prompt-based way (which might not always squeeze the absolute best performance from a model without fine-tuning). The **ease-of-use is moderate** – using the CLI is simple, but customizing anything beyond basics may require editing config files or writing some Python (though documentation exists). Also, because it prioritizes breadth, the evaluation on a large suite of tasks can be time-consuming for big models. In summary, it’s excellent for a research *eval sweep*, but might be overkill if you only care about one or two tasks.

*Links:* [EleutherAI LM Harness GitHub](https://github.com/EleutherAI/lm-evaluation-harness) ([GitHub - EleutherAI/lm-evaluation-harness: A framework for few-shot evaluation of language models.](https://github.com/EleutherAI/lm-evaluation-harness#:~:text=Overview)); Official documentation in the repo (docs/ directory) with task lists and usage. The Hugging Face Open LLM Leaderboard (see below) also provides examples of harness usage for specific tasks.

**Hugging Face LightEval (Unified LLM Benchmarking Toolkit)**

**LightEval** is a relatively new (2024/2025) all-in-one **evaluation framework by Hugging Face** ([LightEval Deep Dive: Hugging Face’s All-in-One Framework for LLM Evaluation - Cohorte Projects](https://www.cohorte.co/blog/lighteval-deep-dive-hugging-faces-all-in-one-framework-for-llm-evaluation" \l ":~:text=LightEval%20is%20an%20open,integrates%20seamlessly%20with%20Hugging%20Face%E2%80%99s)). It was created to streamline the process of evaluating LLMs across many benchmarks and was in fact inspired by and built upon the EleutherAI harness and Stanford’s HELM ([LightEval Deep Dive: Hugging Face’s All-in-One Framework for LLM Evaluation - Cohorte Projects](https://www.cohorte.co/blog/lighteval-deep-dive-hugging-faces-all-in-one-framework-for-llm-evaluation" \l ":~:text=library%20for%20multi,and%20transparency%20in%20one%20framework)). LightEval offers a unified Python API and CLI, integration with the Hugging Face ecosystem (Accelerate, Hub, etc.), and a large library of tasks and metrics.

* **What it Benchmarks:** Like Eleuther’s harness, LightEval supports *hundreds of evaluation tasks* – from classic benchmarks (MMLU, HellaSwag, TruthfulQA, ARC, BIG-bench tasks, GSM8K math, etc.) to custom community-contributed tasks ([LightEval Deep Dive: Hugging Face’s All-in-One Framework for LLM Evaluation - Cohorte Projects](https://www.cohorte.co/blog/lighteval-deep-dive-hugging-faces-all-in-one-framework-for-llm-evaluation" \l ":~:text=,60%2B%20benchmarks%20%28ARC)). It also incorporates scenarios from HELM and the Open LLM Leaderboard. Essentially, it covers **LM-specific benchmarks** extensively (common sense, knowledge, math, coding, etc.), and likely some standard QA or summarization scenarios too. It’s designed to be *holistic*, allowing multiple metrics per task (e.g., accuracy, and also bias metrics if applicable) similar to HELM ([LightEval Deep Dive: Hugging Face’s All-in-One Framework for LLM Evaluation - Cohorte Projects](https://www.cohorte.co/blog/lighteval-deep-dive-hugging-faces-all-in-one-framework-for-llm-evaluation" \l ":~:text=Stanford%20HELM%20Holistic%20eval%20framework,Continuous%20eval%20service)).
* **Supported Models:** It is very flexible – it can evaluate **local Hugging Face models** (single GPU or multi-GPU via Accelerate), distributed setups (via tools like Nanotron), optimized backends like **vLLM** (for high-speed inference) ([LightEval Deep Dive: Hugging Face’s All-in-One Framework for LLM Evaluation - Cohorte Projects](https://www.cohorte.co/blog/lighteval-deep-dive-hugging-faces-all-in-one-framework-for-llm-evaluation" \l ":~:text=,%E2%80%93%20using%20a%20common%20interface)) ([LightEval Deep Dive: Hugging Face’s All-in-One Framework for LLM Evaluation - Cohorte Projects](https://www.cohorte.co/blog/lighteval-deep-dive-hugging-faces-all-in-one-framework-for-llm-evaluation" \l ":~:text=match%20at%20L537%20The%20framework%E2%80%99s,can%20tap%20into%20highly%20optimized)), or even remote models through APIs (Hugging Face Inference API, OpenAI API, etc.) ([LightEval Deep Dive: Hugging Face’s All-in-One Framework for LLM Evaluation - Cohorte Projects](https://www.cohorte.co/blog/lighteval-deep-dive-hugging-faces-all-in-one-framework-for-llm-evaluation" \l ":~:text=,%E2%80%93%20using%20a%20common%20interface)). This means any model you can load with AutoModelForCausalLM is supported, including newer ones like LLaMA 3.2 or DeepSeek (assuming you have the checkpoint), and you can switch to faster engines easily. Integration with Hugging Face Hub is a highlight: you can load a model from the Hub and even have LightEval push the evaluation results to the Hub for sharing.
* **Integration with Hugging Face:** Deep integration. LightEval works with **Transformers** models and **Accelerate** for parallelism ([LightEval Deep Dive: Hugging Face’s All-in-One Framework for LLM Evaluation - Cohorte Projects](https://www.cohorte.co/blog/lighteval-deep-dive-hugging-faces-all-in-one-framework-for-llm-evaluation" \l ":~:text=github,LM%20Evaluation%20Harness%20and%20drew)) ([LightEval Deep Dive: Hugging Face’s All-in-One Framework for LLM Evaluation - Cohorte Projects](https://www.cohorte.co/blog/lighteval-deep-dive-hugging-faces-all-in-one-framework-for-llm-evaluation" \l ":~:text=At%20a%20high%20level%2C%20LightEval%E2%80%99s,consists%20of%20the%20following%20components)). It uses Hugging Face datasets/metrics under the hood and can log outputs to the Hub. You can run evaluations via a high-level API in Python, or via a CLI tool. Because it’s by HF, it understands pretrained model IDs, and can use huggingface’s TextGenerationInference (TGI) server or Accelerate launch for multi-GPU, etc. This makes it very convenient to evaluate models available on the Hub.
* **Example Usage:** Using LightEval’s Python API, you might do something like:
* from lighteval import evaluate\_model, TextGenerationTask
* results = evaluate\_model(
* model="meta-llama/Llama-3.2-1B", # model from HF Hub
* task=TextGenerationTask("hellaswag"), # evaluate on HellaSwag task
* num\_shots=5,
* backend="accelerate", # or 'vllm' for faster inference
* device\_map="auto"
* )
* print(results.metrics) # outputs accuracy, etc.

There is also a CLI interface (lighteval ...) with similar options. The key point is you can specify the model by name and the task by name, and LightEval handles the rest – downloading data, running prompts, computing metrics.

* **Pros:** **User-friendly and unified** – it’s designed to be one toolkit that “just works” for many models and benchmarks ([LightEval Deep Dive: Hugging Face’s All-in-One Framework for LLM Evaluation - Cohorte Projects](https://www.cohorte.co/blog/lighteval-deep-dive-hugging-faces-all-in-one-framework-for-llm-evaluation" \l ":~:text=via%20simple%20Python%20interfaces,Extensible%20with%20new%20task)) ([LightEval Deep Dive: Hugging Face’s All-in-One Framework for LLM Evaluation - Cohorte Projects](https://www.cohorte.co/blog/lighteval-deep-dive-hugging-faces-all-in-one-framework-for-llm-evaluation" \l ":~:text=actively%20developed.%20%5BGitHub%5D%20Very%20user,Extensible%20with%20new%20task)). One command (or function) can evaluate any model on any supported benchmark, which is great for quick comparisons. It has **multi-backend support**, meaning you can easily toggle between speed-optimized backends (like vLLM, which significantly speeds up evaluation of large models ([LightEval Deep Dive: Hugging Face’s All-in-One Framework for LLM Evaluation - Cohorte Projects](https://www.cohorte.co/blog/lighteval-deep-dive-hugging-faces-all-in-one-framework-for-llm-evaluation" \l ":~:text=The%20framework%E2%80%99s%20speed%20optimizations%20and,can%20tap%20into%20highly%20optimized))) and standard PyTorch, without changing the evaluation logic. It’s built and maintained by Hugging Face, so it’s kept up-to-date with new models and datasets, and it integrates with their Hub for result tracking. Also, being open-source and extensible, you can add new tasks or metrics if needed ([LightEval Deep Dive: Hugging Face’s All-in-One Framework for LLM Evaluation - Cohorte Projects](https://www.cohorte.co/blog/lighteval-deep-dive-hugging-faces-all-in-one-framework-for-llm-evaluation" \l ":~:text=,Supports%20HuggingFace%2C%20OpenAI%2C%20vLLM)). In short, it combines the breadth of Eleuther’s harness with the **ease-of-use of Hugging Face’s ecosystem** ([LightEval Deep Dive: Hugging Face’s All-in-One Framework for LLM Evaluation - Cohorte Projects](https://www.cohorte.co/blog/lighteval-deep-dive-hugging-faces-all-in-one-framework-for-llm-evaluation" \l ":~:text=Framework%20Functionality%20Extensibility%20Ease%20of,API%2C%20integrates%20with%20HF%20workflows)).
* **Cons:** As a newer framework, it might still be maturing – there could be occasional rough edges or less community familiarity (compared to the Eleuther harness which many have used). The documentation is improving, but users might need to get used to its configuration style. Another consideration: LightEval is powerful, which also means it has many options; for a beginner, the multitude of backends and tasks might be a bit overwhelming at first (though sane defaults are provided). Performance-wise, while it tries to be efficient, running large-scale evals on big models still requires serious compute – LightEval just makes it easier to orchestrate. There aren’t many downsides beyond these; overall it is poised to become a go-to evaluation toolkit.

*Links:* [LightEval GitHub](https://github.com/huggingface/lighteval) ([LightEval Deep Dive: Hugging Face’s All-in-One Framework for LLM Evaluation - Cohorte Projects](https://www.cohorte.co/blog/lighteval-deep-dive-hugging-faces-all-in-one-framework-for-llm-evaluation" \l ":~:text=github,and%20transparency%20in%20one%20framework)); Hugging Face documentation for LightEval ([LightEval Deep Dive: Hugging Face’s All-in-One Framework for LLM Evaluation - Cohorte Projects](https://www.cohorte.co/blog/lighteval-deep-dive-hugging-faces-all-in-one-framework-for-llm-evaluation" \l ":~:text=,60%2B%20benchmarks%20%28ARC)); a [deep-dive blog post](https://www.cohorte.co/blog/lighteval-deep-dive-hugging-faces-all-in-one-framework-for-llm-evaluation) ([LightEval Deep Dive: Hugging Face’s All-in-One Framework for LLM Evaluation - Cohorte Projects](https://www.cohorte.co/blog/lighteval-deep-dive-hugging-faces-all-in-one-framework-for-llm-evaluation" \l ":~:text=LightEval%20is%20an%20open,integrates%20seamlessly%20with%20Hugging%20Face%E2%80%99s)) comparing it with harness and HELM.

**Stanford HELM (Holistic Evaluation of Language Models)**

**HELM** is an evaluation framework developed by Stanford CRFM focusing on **holistic evaluation** – covering not just accuracy on tasks, but also aspects like bias, fairness, robustness, calibration, etc. ([LightEval Deep Dive: Hugging Face’s All-in-One Framework for LLM Evaluation - Cohorte Projects](https://www.cohorte.co/blog/lighteval-deep-dive-hugging-faces-all-in-one-framework-for-llm-evaluation" \l ":~:text=Stanford%20HELM%20Holistic%20eval%20framework,Continuous%20eval%20service)). It provides a *structured, scenario-based* approach to evaluating language models. HELM is behind a public benchmark (the HELM website/leaderboards) and it has an open-source framework for reproducing those evaluations.

* **What it Benchmarks:** HELM defines **scenarios** (tasks) and evaluates models across a broad spectrum of metrics for each scenario ([LightEval Deep Dive: Hugging Face’s All-in-One Framework for LLM Evaluation - Cohorte Projects](https://www.cohorte.co/blog/lighteval-deep-dive-hugging-faces-all-in-one-framework-for-llm-evaluation" \l ":~:text=Stanford%20HELM%20Holistic%20eval%20framework,Continuous%20eval%20service)). Scenarios include many standard NLP tasks: open-ended QA, reading comprehension, summarization, dialogue, common-sense reasoning, mathematical word problems, and more – around **40+ scenarios** as of its latest version ([LightEval Deep Dive: Hugging Face’s All-in-One Framework for LLM Evaluation - Cohorte Projects](https://www.cohorte.co/blog/lighteval-deep-dive-hugging-faces-all-in-one-framework-for-llm-evaluation" \l ":~:text=evals.%20,Very%20easy%20to%20consume)). For example, it has a scenario for MMLU (knowledge QA), for HellaSwag, for TruthfulQA, but also scenarios for things like “Legal contract QA” or “News article summarization.” For each scenario, HELM often evaluates multiple metrics: e.g., accuracy, but also **bias** (via targeted tests), **toxicity**, or **calibration** of the model’s probabilities. It aims to paint a comprehensive picture of a model’s strengths and weaknesses.
* **Supported Models:** The HELM framework can evaluate any **AutoModelForCausalLM** (for language tasks) from Hugging Face, as well as models accessible via API (they included OpenAI, Cohere, etc., in their leaderboards) ([Hugging Face Model Hub Integration - CRFM HELM](https://crfm-helm.readthedocs.io/en/latest/huggingface_models/#:~:text=Hugging%20Face%20Model%20Hub%20Integration)) ([Hugging Face Model Hub Integration - CRFM HELM](https://crfm-helm.readthedocs.io/en/latest/huggingface_models/#:~:text=HELM%20can%20be%20used%20to,be%20supported%20in%20the%20future)). Specifically, the open-source code supports Hugging Face causal models by either specifying the model in a config or via command-line (helm-run --enable-huggingface-models ...) ([Hugging Face Model Hub Integration - CRFM HELM](https://crfm-helm.readthedocs.io/en/latest/huggingface_models/#:~:text=In%20some%20cases%2C%20you%20can,does%20not%20require%20configuration%20files)). It loads these models similarly to other tools. Note that HELM was primarily developed with very large models in mind (like GPT-3, etc., via API), so it can handle a range of model sizes. New models like LLaMA 3.2 could be integrated by pointing HELM to the Hugging Face model ID ([Hugging Face Model Hub Integration - CRFM HELM](https://crfm-helm.readthedocs.io/en/latest/huggingface_models/#:~:text=Hugging%20Face%20model%20IDs%20to,ID%20as%20the%20model%20name)). (DeepSeek R1 might be too large to run, but in theory if it had an API or was loaded with vLLM, HELM could evaluate it.)
* **Integration with Hugging Face:** HELM’s latest version added **Hugging Face Hub integration**, so you can simply refer to a model by its hub ID in the HELM config and it will use AutoModelForCausalLM to load it ([Hugging Face Model Hub Integration - CRFM HELM](https://crfm-helm.readthedocs.io/en/latest/huggingface_models/#:~:text=Hugging%20Face%20Model%20Hub%20Integration)) ([Hugging Face Model Hub Integration - CRFM HELM](https://crfm-helm.readthedocs.io/en/latest/huggingface_models/#:~:text=In%20some%20cases%2C%20you%20can,does%20not%20require%20configuration%20files)). This makes it fairly straightforward to plug in Hugging Face models. However, using HELM isn’t as simple as a one-line call; it typically involves writing a JSON configuration (called a “run spec”) that specifies which models, scenarios, and metrics to run, then using the helm-run command.
* **Example Usage:** A simplified use-case: say you want to evaluate a model on the *boolq* scenario (Boolean Questions, a reading comprehension task). You could use a command:
* helm-run \
* --scenario boolq \
* --model stanford-crfm/BioMedLM \
* --enable-huggingface-models stanford-crfm/BioMedLM

(This is adapted from HELM docs ([Hugging Face Model Hub Integration - CRFM HELM](https://crfm-helm.readthedocs.io/en/latest/huggingface_models/#:~:text=Hugging%20Face%20model%20IDs%20to,ID%20as%20the%20model%20name)) ([Hugging Face Model Hub Integration - CRFM HELM](https://crfm-helm.readthedocs.io/en/latest/huggingface_models/#:~:text=Example%20model%20on%20Hugging%20Face,Hub))). In practice, you might create a YAML/JSON describing multiple scenarios and models. HELM will produce a structured report with various metrics. Because HELM covers multi-metric evaluation, you might see not just accuracy but also calibration error, etc., in the output.

* **Pros:** Extremely **comprehensive** – HELM’s motto is holistic transparency ([GitHub - stanford-crfm/helm: Holistic Evaluation of Language Models (HELM) is an open source Python framework created by the Center for Research on Foundation Models (CRFM) at Stanford for holistic, reproducible and transparent evaluation of foundation models, including large language models (LLMs) and multimodal models.](https://github.com/stanford-crfm/helm#:~:text=Holistic%20Evaluation%20of%20Language%20Models,LLMs%29%20and%20multimodal%20models)). If you need to evaluate not just performance but also ethical aspects (bias or toxicity) and robustness (e.g., accuracy under input perturbations), HELM is designed for that. It provides a consistent framework that was used to compare many major models in a standardized way (the HELM leaderboard). It supports **multiple models and scenarios in one go**, making large-scale comparisons easier. For researchers or organizations concerned with a model’s broader impacts and reliability, HELM’s built-in metrics and scenarios are valuable.
* **Cons:** **Heavy and complex** – HELM is not as plug-and-play as other tools. The setup and configuration can be involved, and running the full suite of scenarios is computationally expensive ([LightEval Deep Dive: Hugging Face’s All-in-One Framework for LLM Evaluation - Cohorte Projects](https://www.cohorte.co/blog/lighteval-deep-dive-hugging-faces-all-in-one-framework-for-llm-evaluation" \l ":~:text=evals.%20,Very%20easy%20to%20consume)) ([LightEval Deep Dive: Hugging Face’s All-in-One Framework for LLM Evaluation - Cohorte Projects](https://www.cohorte.co/blog/lighteval-deep-dive-hugging-faces-all-in-one-framework-for-llm-evaluation" \l ":~:text=scenarios%2C%20metrics%29,Very%20easy%20to%20consume)). It’s more aimed at serious benchmarking efforts rather than quick tests. The framework’s learning curve is higher; you may need to configure YAML files and understand its schema. Also, extending HELM with new scenarios or metrics, while possible ([LightEval Deep Dive: Hugging Face’s All-in-One Framework for LLM Evaluation - Cohorte Projects](https://www.cohorte.co/blog/lighteval-deep-dive-hugging-faces-all-in-one-framework-for-llm-evaluation" \l ":~:text=Stanford%20HELM%20Holistic%20eval%20framework,Continuous%20eval%20service)), ([LightEval Deep Dive: Hugging Face’s All-in-One Framework for LLM Evaluation - Cohorte Projects](https://www.cohorte.co/blog/lighteval-deep-dive-hugging-faces-all-in-one-framework-for-llm-evaluation" \l ":~:text=evals.%20,Very%20easy%20to%20consume)) is a bit complex (suitable for research teams but not casual users). In summary, HELM excels in breadth and depth of evaluation, but at the cost of simplicity. For many users who just want “did my model get X% on MMLU,” a simpler harness might suffice; HELM is for those who need the **full diagnostic evaluation** of a model.

*Links:* [HELM GitHub](https://github.com/stanford-crfm/helm) ([GitHub - stanford-crfm/helm: Holistic Evaluation of Language Models (HELM) is an open source Python framework created by the Center for Research on Foundation Models (CRFM) at Stanford for holistic, reproducible and transparent evaluation of foundation models, including large language models (LLMs) and multimodal models.](https://github.com/stanford-crfm/helm#:~:text=Holistic%20Evaluation%20of%20Language%20Models,LLMs%29%20and%20multimodal%20models)); [HELM project site](https://crfm.stanford.edu/helm) for scenario definitions and papers.

**OpenAI Evals (Custom Evaluation Framework)**

**OpenAI Evals** is an open-source framework by OpenAI for creating and running **custom evaluations** on language models ([GitHub - openai/evals: Evals is a framework for evaluating LLMs and LLM systems, and an open-source registry of benchmarks.](https://github.com/openai/evals#:~:text=Evals%20provide%20a%20framework%20for,any%20of%20that%20data%20publicly)). While primarily intended to evaluate OpenAI’s own models (like GPT-4 via API), it’s a general system where users can define new evals or use a registry of existing ones. It’s useful if you want to test models on specific criteria or creative tasks that aren’t covered by standard benchmarks.

* **What it Benchmarks:** There is a **registry of evals** provided by OpenAI, which includes things like simple math word problems, logic puzzles, trivia questions, code-generation tests (HumanEval for coding), and more ([GitHub - openai/evals: Evals is a framework for evaluating LLMs and LLM systems, and an open-source registry of benchmarks.](https://github.com/openai/evals#:~:text=Evals%20provide%20a%20framework%20for,any%20of%20that%20data%20publicly)). Many of these are not the classic academic benchmarks, but rather custom-curated tests to probe model behavior in various areas. More importantly, you can **write your own evaluation** – define a prompt and an expected answer or a scoring function – and Evals will run the model and measure success rates. Essentially, it benchmarks whatever you define, be it qualitative instruction-following tasks or quantitative QA. (GLUE or MMLU are not pre-built in OpenAI Evals; it’s more for bespoke evaluations or harnessing user-contributed evals.)
* **Supported Models:** Out-of-the-box, it supports models accessible via the OpenAI API (like gpt-3.5, gpt-4). However, since it’s open-source, people have extended it to local models by writing a custom model adapter that wraps a Hugging Face model to mimic the OpenAI API interface. This requires some coding – it’s not a built-in feature. So, *officially*, it’s geared towards OpenAI’s models, but **the framework can support Hugging Face models** if you integrate them. Newer models (LLaMA etc.) could be evaluated if you plug them in as an OpenAI API compatible class. This is non-trivial but possible.
* **Integration with Hugging Face:** There’s no native Hugging Face integration in OpenAI Evals. To use a Hugging Face AutoModelForCausalLM, you would likely have to implement a “model provider” within the Evals framework that calls your model for completions. Most users use Evals with OpenAI’s API key (which won’t apply for local models unless using something like OpenAI’s function calling format).
* **Example Usage:** If using OpenAI’s models, you might run an eval with the CLI after installing:
* openai eval run custom\_math\_eval --model gpt-4

where custom\_math\_eval is an eval defined in the registry. It will output success metrics like “GPT-4 got 85% of these custom math problems correct.” For a Hugging Face model, hypothetically you’d do:

openai eval run custom\_math\_eval --model local:huggingface/my-model

after registering a local:huggingface model handle in the framework (this is illustrative – actual steps involve coding).

* **Pros:** **Highly flexible** – you can evaluate **exactly the behavior you care about**, which is great for targeted testing (e.g., does the model follow a specific style guideline, or how often does it refuse disallowed requests, etc.). It’s collaborative: there’s a growing open-source repository of evals, meaning you might find community-contributed tests for common pitfalls. If you have specific pass/fail criteria or non-standard tasks, Evals can handle them when typical benchmark suites cannot. It’s also up-to-date, being maintained alongside new OpenAI models, and through the API it can evaluate very large models easily (offloading compute to OpenAI’s cloud).
* **Cons:** For local model evaluation, it’s **not straightforward**. The framework expects an API key and queries OpenAI endpoints by default ([GitHub - openai/evals: Evals is a framework for evaluating LLMs and LLM systems, and an open-source registry of benchmarks.](https://github.com/openai/evals#:~:text=Setup)). Adapting it to local models requires engineering. Also, Evals is oriented around *discrete question-answer or checkable tasks* – it’s not for computing perplexity or standard metrics like BLEU. Instead, you often have to provide a ground-truth or a programmatic evaluator for each sample. This is powerful but means more work per task. In short, if your goal is standard benchmarks (which already have evaluators), using Eleuther or LightEval is easier. OpenAI Evals shines when evaluating something unique or doing error analysis on specific scenarios. Additionally, if you want to run large numbers of evals on big models, the cost (if using OpenAI API) or complexity (if hooking up local models) can be a limiting factor.

*Links:* [OpenAI Evals GitHub](https://github.com/openai/evals) ([GitHub - openai/evals: Evals is a framework for evaluating LLMs and LLM systems, and an open-source registry of benchmarks.](https://github.com/openai/evals#:~:text=Evals%20provide%20a%20framework%20for,any%20of%20that%20data%20publicly)); official OpenAI guide for Evals.

**Hugging Face Open LLM Leaderboard (Public Benchmarking Platform)**

The **Open LLM Leaderboard** by Hugging Face is a platform that continuously evaluates submitted models on a fixed suite of benchmarks and reports their scores. While not a library that you install, it’s worth mentioning because it uses some of the above tools under the hood and you can mimic its evaluations. The leaderboard currently tests models on **6 benchmarks**: ARC (Challenge), HellaSwag, MMLU, TruthfulQA, WinoGrande, and GSM8K (math) ([LightEval Deep Dive: Hugging Face’s All-in-One Framework for LLM Evaluation - Cohorte Projects](https://www.cohorte.co/blog/lighteval-deep-dive-hugging-faces-all-in-one-framework-for-llm-evaluation" \l ":~:text=HF%20Open%20LLM%20Leaderboard%20Public,Leaderboard)). It uses a standardized prompt and few-shot setting for each. The maintainers rely on the EleutherAI harness / LightEval to produce those results ([LightEval Deep Dive: Hugging Face’s All-in-One Framework for LLM Evaluation - Cohorte Projects](https://www.cohorte.co/blog/lighteval-deep-dive-hugging-faces-all-in-one-framework-for-llm-evaluation" \l ":~:text=EleutherAI%20LM%20Harness%20Python,framework%20with%20wide%20scenario%2Fmetric%20coverage)) ([LightEval Deep Dive: Hugging Face’s All-in-One Framework for LLM Evaluation - Cohorte Projects](https://www.cohorte.co/blog/lighteval-deep-dive-hugging-faces-all-in-one-framework-for-llm-evaluation" \l ":~:text=HF%20Open%20LLM%20Leaderboard%20Public,Leaderboard)).

* **What it Benchmarks:** Exactly six tasks: four are *knowledge/reasoning benchmarks* (ARC, HellaSwag, MMLU, TruthfulQA, Winogrande) and one is a *math/code* task (GSM8K for math word problems) ([LightEval Deep Dive: Hugging Face’s All-in-One Framework for LLM Evaluation - Cohorte Projects](https://www.cohorte.co/blog/lighteval-deep-dive-hugging-faces-all-in-one-framework-for-llm-evaluation" \l ":~:text=HF%20Open%20LLM%20Leaderboard%20Public,Leaderboard)). These are all evaluated in a few-shot context (e.g., 5-shot for MMLU).
* **Supported Models:** Any model on Hugging Face Hub can be submitted to the leaderboard (if it meets the size/license criteria). They particularly feature open models like LLaMA derivatives, Falcon, etc. If you want to evaluate your model on the same suite, you’d need to have it in HF format.
* **Integration with Hugging Face:** The evaluations are run by HF using their infrastructure. For personal use, you can use the EleutherAI harness or LightEval with the corresponding task prompts (the HF docs provide command examples). In fact, one can reproduce leaderboard results by running EleutherAI’s harness with the specified tasks and few-shot count ([open-llm-leaderboard/open\_llm\_leaderboard - Hugging Face](https://huggingface.co/spaces/open-llm-leaderboard/open_llm_leaderboard/discussions/60#:~:text=open,harness)) ([LightEval Deep Dive: Hugging Face’s All-in-One Framework for LLM Evaluation - Cohorte Projects](https://www.cohorte.co/blog/lighteval-deep-dive-hugging-faces-all-in-one-framework-for-llm-evaluation" \l ":~:text=EleutherAI%20LM%20Harness%20Python,GitHub)). No direct “library” to pip install here – it’s a set of evaluations done for you.
* **Example Usage:** If you want to locally replicate, for example, the leaderboard evaluation of a model on those tasks, you could run:
* lm\_eval --model hf \
* --model\_args pretrained=<your-model>,device=cuda:0 \
* --tasks arc\_challenge,hellaswag,mmlu,truthfulqa\_winogrande,gsm8k \
* --num\_fewshot 5

(This is illustrative; check HF’s leaderboard documentation for the exact task identifiers and few-shot settings.)

* **Pros:** As a **reference point**, the leaderboard is great – you can see how your model would rank and get a quick sense of performance across popular benchmarks. It’s continuously updated and uses consistent evaluation methodology, so it’s a reliable comparison. The fact that it’s integrated with HF Hub means you can trigger evaluations by just submitting your model, no need to run anything yourself.
* **Cons:** Limited to the fixed benchmarks – it doesn’t help if you need other tasks evaluated. Not interactive as a library. And to use it, you have to submit your model (which might not be possible for private models or those not meeting submission criteria). For offline use, you’re back to using the harness or LightEval manually. Essentially, it’s not a tool you import, but a service; we mention it for completeness and to highlight that it leverages the above tools (so any method that works for those 6 benchmarks likely aligns with the leaderboard methodology).

*Links:* [Open LLM Leaderboard page](https://huggingface.co/leaderboards/open-llm-leaderboard) and [GitHub discussion](https://huggingface.co/spaces/HuggingFaceH4/open_llm_leaderboard) (includes info on evaluation setup).

**Speed and Efficiency Benchmarking Tools**

In addition to accuracy and task performance, you often want to measure a model’s **inference speed, latency, and memory usage**. The tools below help benchmark those aspects for Hugging Face-loaded models:

**Transformers Benchmark Utility (PyTorch)**

The Hugging Face *Transformers* library includes a benchmarking utility (though note: it was marked as deprecated in recent docs ([Benchmarks](https://huggingface.co/docs/transformers/v4.47.1/benchmarks#:~:text=Benchmarks)), it’s still usable). The **PyTorchBenchmark** and **TensorFlowBenchmark** classes allow you to measure runtime and memory for forward passes on models ([Benchmarks](https://huggingface.co/docs/transformers/v4.47.1/benchmarks#:~:text=The%20classes%20,for%20both%20inference%20and%20training)) ([Benchmarks](https://huggingface.co/docs/transformers/v4.47.1/benchmarks#:~:text=python%20examples%2Fpytorch%2Fbenchmarking%2Frun_benchmark.py%20)).

* **What it Measures:** **Inference time** (and training time) for a single forward pass under specified conditions, as well as **memory utilization** (peak memory usage) ([Benchmarks](https://huggingface.co/docs/transformers/v4.47.1/benchmarks#:~:text=The%20classes%20,for%20both%20inference%20and%20training)). You can test different batch sizes and sequence lengths. For example, “how many milliseconds and how much GPU RAM does it take for LLaMA-3B to process a batch of 8 with 512 tokens each?” These classes will run the model and record that.
* **Supported Models:** Any model supported by Transformers (so, yes, LLaMA 3.2, etc. if loaded via AutoModel). There are separate classes for PyTorch vs TensorFlow. Most causal LMs are in PyTorch, so PyTorchBenchmark is used. It will handle CPU or GPU (you can specify device).
* **Integration:** This is part of the transformers library. You instantiate PyTorchBenchmarkArguments with details (model names, batch sizes, seq lengths) and then PyTorchBenchmark to run ([Benchmarks](https://huggingface.co/docs/transformers/v4.47.1/benchmarks#:~:text=,PyTorchBenchmarkArguments)) ([Benchmarks](https://huggingface.co/docs/transformers/v4.47.1/benchmarks#:~:text=,TensorFlowBenchmarkArguments)). It will automatically load the model(s) from the Hub if given names. Integration is seamless if your model is on the Hub or local path – just provide the identifier.
* **Example Usage:**
* from transformers import PyTorchBenchmark, PyTorchBenchmarkArguments
* args = PyTorchBenchmarkArguments(
* models=["meta-llama/Llama-3.2-1B"],
* batch\_sizes=[1, 8],
* sequence\_lengths=[128, 512]
* )
* benchmark = PyTorchBenchmark(args)
* results = benchmark.run()
* print(results)

The results will contain timing (seconds per batch) and memory stats for each combination ([Benchmarks](https://huggingface.co/docs/transformers/v4.47.1/benchmarks#:~:text=python%20examples%2Fpytorch%2Fbenchmarking%2Frun_benchmark.py%20)) ([Benchmarks](https://huggingface.co/docs/transformers/v4.47.1/benchmarks#:~:text=,RESULT)). The output might show something like: for seq length 128, batch 8, the model runs at X tokens/sec and uses Y GB GPU memory.

* **Pros:** Very **easy to set up** within code. It’s useful for getting a quick sense of how a model scales with sequence length or batch size. It can compare multiple models if you list several. It provides **both speed and memory metrics**, which is important for efficiency evaluation ([Benchmarks](https://huggingface.co/docs/transformers/v4.47.1/benchmarks#:~:text=The%20classes%20,for%20both%20inference%20and%20training)). No extra installation needed if you have transformers. Great for **regression testing** (e.g., did my optimization improve speed?).
* **Cons:** As noted, Hugging Face has deprecated this in favor of external tools, which implies it might not get new features or could be removed eventually ([Benchmarks](https://huggingface.co/docs/transformers/v4.47.1/benchmarks#:~:text=Benchmarks)). It’s also somewhat limited in scope – it measures end-to-end forward pass, but doesn’t break down where time is spent (for that you’d use profilers). It doesn’t directly measure generation throughput (though you can approximate by using long sequence lengths or looping multiple forwards). For very large models, the benchmark might itself introduce overhead (or be unable to run if model doesn’t fit on one GPU). Despite being simple, it’s not as extensive as some deep-learning benchmarking suites.

*Links:* HF Transformers documentation on [benchmarking utilities](https://huggingface.co/docs/transformers/benchmarks) ([Benchmarks](https://huggingface.co/docs/transformers/v4.47.1/benchmarks#:~:text=The%20classes%20,for%20both%20inference%20and%20training)).

**DeepSpeed and DeepSpeed-MII**

**DeepSpeed** is an optimization library by Microsoft, known for training large models efficiently, but it also has an inference module and a tool called **MII (Massively Intelligent Inference)** for serving models. DeepSpeed’s inference engine focuses on accelerating transformer model inference and reducing memory footprint via techniques like optimized kernels, quantization, and parallelism ([DeepSpeed: Accelerating large-scale model inference and training ...](https://www.microsoft.com/en-us/research/blog/deepspeed-accelerating-large-scale-model-inference-and-training-via-system-optimizations-and-compression/" \l ":~:text=,when%20compared%20with%20existing%20work)).

* **What it Measures/Improves:** DeepSpeed Inference can be used to measure **latency and throughput** under optimized conditions. It’s not a benchmarking *suite* per se, but by using it to serve a model, you can gather performance numbers. DeepSpeed has reported up to **1.9–4.4× lower latency and 3.4–6.2× higher throughput** for LLM inference compared to baseline HF transformers ([DeepSpeed: Accelerating large-scale model inference and training ...](https://www.microsoft.com/en-us/research/blog/deepspeed-accelerating-large-scale-model-inference-and-training-via-system-optimizations-and-compression/" \l ":~:text=DeepSpeed%3A%20Accelerating%20large,when%20compared%20with%20existing%20work)). It also reduces memory usage (through tensor parallelism and partial offloading).
* **Supported Models:** It supports most transformer models (via integration with Hugging Face). You can load a HF model into DeepSpeed with few lines of code. It’s especially useful for very large models that need tensor parallelism across GPUs (like DeepSeek R1, if one attempted to run it, DeepSpeed would be one of the only ways).
* **Integration:** Through DeepSpeed’s wrapper or the MII library. For example, using MII:
* import mii
* mii.deploy(task='text-generation', model="meta-llama/Llama-3.2-1B", deployment\_name="llama32", enable\_deepspeed=True)

This sets up a local server with the model optimized by DeepSpeed. You can then use mii.query("llama32", {"text": "Hello"}) to measure response time. Under the hood it uses DeepSpeed and you can see logs of speed.

* **Example Usage:** If you just want to time with DeepSpeed without serving, one can do:
* import deepspeed
* model = AutoModelForCausalLM.from\_pretrained(..., torch\_dtype=torch.float16).half()
* ds\_model = deepspeed.init\_inference(model, dtype=torch.float16, mp\_size=1) # initialize optimized model
* # Warmup
* input\_ids = tokenizer("Hello world", return\_tensors='pt').input\_ids.cuda()
* \_ = ds\_model(input\_ids)
* torch.cuda.synchronize()
* start = time.time()
* \_ = ds\_model(input\_ids)
* torch.cuda.synchronize()
* print("Latency:", time.time() - start)

This would give you a latency for one forward pass with DeepSpeed optimizations.

* **Pros:** **Significant speed-ups** and memory savings for large models – if you care about maximizing throughput or squeezing a model onto limited hardware, DeepSpeed is very effective (quantization and kernel fusion can give multi-fold improvements ([DeepSpeed: Accelerating large-scale model inference and training ...](https://www.microsoft.com/en-us/research/blog/deepspeed-accelerating-large-scale-model-inference-and-training-via-system-optimizations-and-compression/" \l ":~:text=,when%20compared%20with%20existing%20work))). It’s one of the few tools that can make **giant models feasible** by sharding across GPUs. For benchmarking, if you want to compare “baseline vs optimized” performance, DeepSpeed provides the optimized side. It’s also fairly well-maintained and has industry adoption.
* **Cons:** It’s a bit **complex to set up** if you’re not familiar. The gains are more evident on very large models or high-throughput scenarios; for smaller models (a few billion parameters), the overhead might not be worth it. DeepSpeed’s focus is on optimization rather than providing a nice report – so you have to do the timing measurements manually or through its logs. Also, it’s primarily for PyTorch; if you need multi-platform or a simpler interface, you might consider alternatives like ONNX Runtime or TensorRT. Finally, using DeepSpeed may involve installing custom kernels or matching CUDA versions, which can be an added hassle for quick experiments.

*Links:* [DeepSpeed Inference Blog](https://www.microsoft.com/en-us/research/blog/deepspeed-extreme-scale-model-inference-speedup/) (Microsoft) – showcasing speedups ([DeepSpeed: Accelerating large-scale model inference and training ...](https://www.microsoft.com/en-us/research/blog/deepspeed-accelerating-large-scale-model-inference-and-training-via-system-optimizations-and-compression/" \l ":~:text=,when%20compared%20with%20existing%20work)); [DeepSpeed MII GitHub](https://github.com/microsoft/DeepSpeed-MII).

**Text Generation Inference (TGI)**

**TGI** (formerly Hugging Face Accelerate Inference) is a high-performance inference server for text generation models. It’s not exactly a benchmarking library, but a serving stack that is highly optimized (uses bitsandbytes int8, chunks KV cache across GPUs, etc.) to achieve high throughput. If you spin up a model on TGI, you can observe its throughput (tokens/sec) under real conditions.

* **What it Measures:** By running a model on TGI and hitting it with requests, you can measure **end-to-end throughput, latency under load**, and memory usage on the server. TGI is designed to handle multiple concurrent requests and large batch generation efficiently.
* **Supported Models:** All Hugging Face AutoModelForCausalLM models (TGI is model-agnostic, but you need enough GPU memory for the model or use sharding). It’s used for models like LLaMA-65B or Falcon on multiple GPUs.
* **Integration:** You launch TGI either via Docker or text-generation-launcher. For example:
* text-generation-launcher --model-id meta-llama/Llama-3.2-1B --num-shard 1

This will start a server. You can then use an API or the text-generation Python client to send prompts and measure how fast responses come back. It integrates with HF pipelines as well (there’s a pipeline(..., model="http://localhost:80") possibility).

* **Example Usage:** After starting the server as above, you could do:
* import requests, time
* prompt = "Once upon a time,"
* t0 = time.time()
* response = requests.post("http://localhost:80/generate", json={"inputs": prompt, "max\_new\_tokens": 100})
* dt = time.time() - t0
* tokens\_generated = len(response.json()[0]['generated\_text'].split())
* print(f"Generated {tokens\_generated} tokens in {dt:.2f}s, throughput {tokens\_generated/dt:.2f} tokens/sec")

This gives an idea of inference speed.

* **Pros:** TGI is **highly optimized for real-world use** – it can be more efficient than naive Python loops, especially when serving multiple requests. It’s used in production deployments, meaning it’s stable and can handle large models (with sharding). For benchmarking, it’s useful to simulate a deployed environment and see how a model performs over time or under concurrency. It also supports features like streaming, which can affect perceived latency.
* **Cons:** It’s more of a deployment tool than a simple benchmark script. Setting it up (especially multi-GPU) requires Docker or Kubernetes knowledge. If you just want quick numbers on a single forward pass, the HF Benchmark or a simple loop might be easier. TGI shines when measuring at scale – if that’s not your need, it might be overkill. Additionally, interpreting results requires designing your test (number of requests, prompt lengths, etc.), so it’s not as straightforward as a one-command benchmark.

*Links:* [TGI GitHub](https://github.com/huggingface/text-generation-inference) and [Hugging Face Inference documentation](https://huggingface.co/docs/text-generation-inference).

**Other Considerations for Speed Tests**

Beyond the above, you can always do manual benchmarking using Python’s time and memory profiling tools. For example, using **PyTorch CUDA timing** and **memory tracking** (via torch.cuda.max\_memory\_allocated() or the **memory\_profiler** package) can give fine-grained performance insights. Frameworks like **TensorRT** (NVIDIA) or **ONNX Runtime** have their own benchmarking utilities if you convert the model – they can yield huge speedups for supported models, but integration effort is higher.

In summary, for **perplexity and NLP task evaluations**, tools like *Hugging Face Evaluate* (for metrics), *jiant* (GLUE/SQuAD), and especially *LM Evaluation Harness* or *LightEval* (few-shot benchmarks like MMLU, HellaSwag, etc.) are very useful. For **comprehensive multi-metric evaluation**, Stanford’s HELM is available. For **efficiency benchmarking**, the Transformers built-in benchmark or performance-focused libraries like DeepSpeed and TGI can help measure and improve inference speed and resource usage. All these tools support Hugging Face models one way or another, making them suitable for evaluating modern models like LLaMA 3.2 or DeepSeek R1 in 2025.

**Sources:** The information above is based on official documentation and recent analyses of each tool, including Hugging Face docs ( [Evaluate](https://huggingface.co/docs/evaluate/en/index#:~:text=A%20library%20for%20easily%20evaluating,machine%20learning%20models%20and%20datasets)) ([Using the evaluator](https://huggingface.co/docs/evaluate/en/base_evaluator#:~:text=Currently%20supported%20tasks%20are%3A)), EleutherAI’s repository ([GitHub - EleutherAI/lm-evaluation-harness: A framework for few-shot evaluation of language models.](https://github.com/EleutherAI/lm-evaluation-harness#:~:text=Features%3A)) ([LightEval Deep Dive: Hugging Face’s All-in-One Framework for LLM Evaluation - Cohorte Projects](https://www.cohorte.co/blog/lighteval-deep-dive-hugging-faces-all-in-one-framework-for-llm-evaluation" \l ":~:text=EleutherAI%20LM%20Harness%20Python,framework%20with%20wide%20scenario%2Fmetric%20coverage)), a LightEval deep-dive blog ([LightEval Deep Dive: Hugging Face’s All-in-One Framework for LLM Evaluation - Cohorte Projects](https://www.cohorte.co/blog/lighteval-deep-dive-hugging-faces-all-in-one-framework-for-llm-evaluation" \l ":~:text=Framework%20Functionality%20Extensibility%20Ease%20of,API%2C%20integrates%20with%20HF%20workflows)) ([LightEval Deep Dive: Hugging Face’s All-in-One Framework for LLM Evaluation - Cohorte Projects](https://www.cohorte.co/blog/lighteval-deep-dive-hugging-faces-all-in-one-framework-for-llm-evaluation" \l ":~:text=,60%2B%20benchmarks%20%28ARC)), Stanford’s HELM reports ([LightEval Deep Dive: Hugging Face’s All-in-One Framework for LLM Evaluation - Cohorte Projects](https://www.cohorte.co/blog/lighteval-deep-dive-hugging-faces-all-in-one-framework-for-llm-evaluation" \l ":~:text=Stanford%20HELM%20Holistic%20eval%20framework,Continuous%20eval%20service)), and performance reports from Hugging Face and Microsoft ([Benchmarks](https://huggingface.co/docs/transformers/v4.47.1/benchmarks#:~:text=The%20classes%20,for%20both%20inference%20and%20training)) ([DeepSpeed: Accelerating large-scale model inference and training ...](https://www.microsoft.com/en-us/research/blog/deepspeed-accelerating-large-scale-model-inference-and-training-via-system-optimizations-and-compression/" \l ":~:text=,when%20compared%20with%20existing%20work)).