

# H2O LLM Studio | Documentation

A framework and no-code GUI designed for fine-tuning state-of-the-art large language models (LLMs)

## Contents

<b>What is H2O LLM Studio?</b>	<b>5</b>
Who is H2O LLM Studio for? . . . . .	5
<b>Set up H2O LLM Studio</b>	<b>6</b>
Prerequisites . . . . .	6
Installation . . . . .	6
Linux/Ubuntu installation (recommended) . . . . .	6
Using the requirements.txt file . . . . .	7
Windows installation . . . . .	7
Install custom package . . . . .	8
Run H2O LLM Studio . . . . .	8
Run H2O LLM Studio GUI . . . . .	8
Run using Docker from a nightly build . . . . .	9
Run by building your own Docker image . . . . .	10
Run with command line interface (CLI) . . . . .	10
<b>H2O LLM Studio performance</b>	<b>11</b>
<b>Model flow</b>	<b>15</b>
Step 1: Import a dataset . . . . .	15
Step 2: Create an experiment . . . . .	15
Step 3: Monitor an experiment . . . . .	15
Step 4: Compare experiments . . . . .	15
Step 5: Export a model to Hugging Face Hub . . . . .	15
<b>Core features</b>	<b>16</b>
No-code fine-tuning . . . . .	16
Highly customizable (wide range of hyperparameters) . . . . .	16
Advanced evaluation metrics and experiment comparison . . . . .	16
Instant publishing models . . . . .	16
Instant feedback on model performance . . . . .	16
<b>Concepts</b>	<b>17</b>
LLM . . . . .	17
Parameters and hyperparameters . . . . .	17
LLM Backbone . . . . .	17
Generative AI . . . . .	17
Foundation model . . . . .	17
Fine-tuning . . . . .	17
LoRA (Low-Rank Adaptation) . . . . .	17
Quantization . . . . .	18
8-bit model training with a low memory footprint . . . . .	18
BLEU . . . . .	18
Perplexity . . . . .	18

<b>Supported data connectors and format</b>	<b>19</b>
Data connectors . . . . .	19
Data format . . . . .	19
Example data . . . . .	19
<b>Import a dataset</b>	<b>20</b>
Import data . . . . .	20
Data Sources . . . . .	20
Configure dataset . . . . .	23
Data validity check . . . . .	24
View dataset . . . . .	24
<b>View and manage dataset</b>	<b>26</b>
View a dataset . . . . .	26
Dataset tabs . . . . .	26
Edit a dataset . . . . .	27
Delete a dataset . . . . .	27
<b>Merge datasets</b>	<b>28</b>
<b>Experiment settings</b>	<b>29</b>
General settings . . . . .	29
Dataset . . . . .	29
Problem type . . . . .	29
Import config from YAML . . . . .	29
Experiment name . . . . .	29
LLM backbone . . . . .	29
Dataset settings . . . . .	29
Train dataframe . . . . .	29
Validation strategy . . . . .	30
Validation size . . . . .	30
Data sample . . . . .	30
System column . . . . .	30
Prompt column . . . . .	30
Prompt column separator . . . . .	30
Answer column . . . . .	30
Parent ID column . . . . .	30
Text prompt start . . . . .	30
Text answer separator . . . . .	30
Add EOS token to prompt . . . . .	30
Add EOS token to answer . . . . .	30
Mask prompt labels . . . . .	31
Tokenizer settings . . . . .	31
Max length . . . . .	31
Add prompt answer tokens . . . . .	31
Padding quantile . . . . .	31
Architecture settings . . . . .	31
Backbone Dtype . . . . .	31
Gradient Checkpointing . . . . .	31
Intermediate dropout . . . . .	31
Pretrained weights . . . . .	31
Training settings . . . . .	32
Loss function . . . . .	32
Optimizer . . . . .	32
Learning rate . . . . .	32
Differential learning rate layers . . . . .	32
Freeze layers . . . . .	32
Use Flash Attention 2 . . . . .	32
Batch size . . . . .	33

Epochs . . . . .	33
Schedule . . . . .	33
Warmup epochs . . . . .	33
Weight decay . . . . .	33
Gradient clip . . . . .	33
Grad accumulation . . . . .	33
Lora . . . . .	33
Use Dora . . . . .	34
Lora R . . . . .	34
Lora Alpha . . . . .	34
Lora dropout . . . . .	34
Lora target modules . . . . .	34
Lora unfreeze layers . . . . .	34
Save checkpoint . . . . .	34
Evaluation epochs . . . . .	34
Evaluate before training . . . . .	34
Train validation data . . . . .	35
Augmentation settings . . . . .	35
Token mask probability . . . . .	35
Skip parent probability . . . . .	35
Random parent probability . . . . .	35
Neptune noise alpha . . . . .	35
Prediction settings . . . . .	35
Metric . . . . .	35
Metric GPT model . . . . .	35
Metric GPT template . . . . .	35
Min length inference . . . . .	36
Max length inference . . . . .	36
Batch size inference . . . . .	36
Do sample . . . . .	36
Num beams . . . . .	36
Temperature . . . . .	36
Repetition penalty . . . . .	36
Stop tokens . . . . .	36
Top K . . . . .	36
Top P . . . . .	36
Environment settings . . . . .	36
GPUs . . . . .	36
Mixed precision . . . . .	37
Compile model . . . . .	37
Find unused parameters . . . . .	37
Trust remote code . . . . .	37
Huggingface branch . . . . .	37
Number of workers . . . . .	37
Seed . . . . .	37
Logging settings . . . . .	37
Logger . . . . .	37
Neptune project . . . . .	37
<b>Create an experiment</b>	<b>38</b>
Run an experiment on the OASST data via CLI . . . . .	39
<b>View and manage experiments</b>	<b>40</b>
View an experiment . . . . .	40
Experiment tabs . . . . .	40
Stop an experiment . . . . .	43
Delete an experiment . . . . .	44
<b>Compare experiments</b>	<b>45</b>

<b>Publish model to HuggingFace</b>	<b>46</b>
Download a model . . . . .	46
<b>Evaluate model using an AI judge</b>	<b>48</b>
<b>Import a model to h2oGPT</b>	<b>50</b>
Steps . . . . .	50
Examples: . . . . .	50
<b>FAQs</b>	<b>51</b>
What are the general recommendations for using H2O LLM Studio? . . . . .	51
Is the tool multi-user or single user? . . . . .	51
How can human feedback be applied in LLM Studio? . . . . .	51
How does H2O LLM Studio evaluate the fine-tuned model? . . . . .	51
Can I use a different AI Judge than ChatGPT? . . . . .	51
How much data is generally required to fine-tune a model? . . . . .	51
Are there any recommendations for which backbone to use? Are some backbones better for certain types of tasks? . . . . .	52
What if my data is not in question-and-answer form and I just have documents? How can I fine-tune the LLM model? . . . . .	52
Can the adapter be downloaded after fine-tuning so that the adapter can be combined with the backbone LLM for deployment? . . . . .	52
I encounter GPU out-of-memory issues. What can I change to be able to train large models? . . . . .	52
When does the model stop the fine-tuning process? . . . . .	52
What is the maximum dataset size that an LLM Studio instance can handle? . . . . .	52
Where does H2O LLM Studio store its data? . . . . .	53
How can I update H2O LLM Studio? . . . . .	53
Once I have the LoRA, what is the recommended way of utilizing it with the base model? . . . . .	53
How to use H2O LLM Studio in Windows? . . . . .	53
How can I easily fine-tune a large language model (LLM) using the command-line interface (CLI) of H2O LLM Studio when I have limited GPU memory? . . . . .	53
Can I run a validation metric on a model post-training, optionally on a different validation dataset? . . . . .	53
What are the hardware/infrastructure sizing recommendations for H2O LLM Studio? . . . . .	54
<b>Key terms</b>	<b>55</b>
Prompt Engineering . . . . .	55
Agents . . . . .	55
ELO . . . . .	55
Vector Store . . . . .	55
Pre-training . . . . .	55
Attention . . . . .	55
Embedding . . . . .	55
Language Model . . . . .	55
Transformer . . . . .	55
Encoders and Decoders . . . . .	55
Text generation . . . . .	55
In-context learning . . . . .	55
Few-shot learning . . . . .	56
Summarization . . . . .	56
Fine-tuning . . . . .	56
GPT . . . . .	56
<b>GPU deployment</b>	<b>56</b>
<b>Tokenization</b>	<b>56</b>

## What is H2O LLM Studio?

H2O LLM Studio is an open-source, no-code [LLM](#) graphical user interface (GUI) designed for fine-tuning state-of-the-art large language models.

[Fine-tuning](#) a pretrained language model requires coding expertise and extensive knowledge about the model and its [hyperparameters](#), however H2O LLM Studio enables NLP practitioners to fine-tune their LLMs easily with no need for coding and better flexibility over customization.

H2O LLM Studio also lets you chat with the fine-tuned model and receive instant feedback about model performance.

## Who is H2O LLM Studio for?

H2O LLM Studio is a free and open-source tool that is designed for anyone who wants to fine-tune their own language models. It is designed to be easy to use and accessible to everyone regardless of their technical expertise.

NLP practitioners and data scientists in particular may find it useful to easily and effectively create and fine-tune large language models.

# Set up H2O LLM Studio

This page guides you through setting up and installing H2O LLM Studio on your local system.

First, download the H2O LLM Studio package from the [H2O LLM Studio Github repository](#). You can use `git clone` or navigate to the [releases page](#) and download the `.zip` file found within the **Assets** of the relevant release.

## Prerequisites

H2O LLM Studio requires the following minimum requirements:

- A machine with Ubuntu 16.04+ with atleast one recent Nvidia GPU
- Have at least 128GB+ of system RAM. Larger models and complex tasks may require 256GB+ or more.
- Nvidia drivers v470.57.02 or a later version
- Access to the following URLs:
  - `developer.download.nvidia.com`
  - `pypi.org`
  - `huggingface.co`
  - `download.pytorch.org`
  - `cdn-lfs.huggingface.co`

### Notes:

- Atleast 24GB of GPU memory is recommended for larger models.
- For more information on performance benchmarks based on the hardware setup, see [H2O LLM Studio performance](#).
- The required URLs are accessible by default when you start a GCP instance, however, if you have network rules or custom firewalls in place, it is recommended to confirm that the URLs are accessible before running `make setup`.

## Installation

### Linux/Ubuntu installation (recommended)

The recommended way to install H2O LLM Studio is using pipenv with Python 3.10. To install Python 3.10 on Ubuntu 16.04+, execute the following commands.

#### System installs (Python 3.10)

```
sudo add-apt-repository ppa:deadsnakes/ppa
sudo apt install python3.10
sudo apt-get install python3.10-distutils
curl -sS https://bootstrap.pypa.io/get-pip.py | python3.10
```

#### Install NVIDIA drivers (if required)

If you are deploying on a ‘bare metal’ machine running Ubuntu, you may need to install the required Nvidia drivers and CUDA. The following commands show how to retrieve the latest drivers for a machine running Ubuntu 20.04 as an example. You can update the following based on your respective operating system.

`wget`

```
https://developer.download.nvidia.com/compute/cuda/repos/ubuntu2004/x86_64/cuda-ubuntu2004.pin{" " }
```

```
sudo mv cuda-ubuntu2004.pin
/etc/apt/preferences.d/cuda-repository-pin-600
```

`wget`

```
https://developer.download.nvidia.com/compute/cuda/11.4.3/local_installers/cuda-repo-ubuntu2004-11-4-local_11
```

```
sudo dpkg -i
cuda-repo-ubuntu2004-11-4-local_11.4.3-470.82.01-1_amd64.deb
sudo apt-key add /var/cuda-repo-ubuntu2004-11-4-local/7fa2af80.pub
sudo apt-get -y update
sudo apt-get -y install cuda
```

#### Create virtual environment (pipenv)

The following command creates a virtual environment using pipenv and will install the dependencies using pipenv.

```
make setup
```

### Using the requirements.txt file

If you wish to use conda or another virtual environment, you can also install the dependencies using the `requirements.txt` file.

```
pip install -r requirements.txt
```

### Windows installation

Follow the steps below to install H2O LLM Studio on a Windows machine using Windows Subsystem for Linux [WSL2](#).

1. Download the [latest nvidia driver](#) for Windows.
2. Open PowerShell or a Windows Command Prompt window in administrator mode.
3. Run the following command to confirm that the driver is installed properly and see the driver version.

```
nvidia-smi
```

4. Run the following command to install WSL2.

```
wsl --install
```

5. Launch the WSL2 Ubuntu installation.
6. Install the WSL2 Nvidia CUDA Drivers.

```
wget
```

```
https://developer.download.nvidia.com/compute/cuda/repos/wsl-ubuntu/x86_64/cuda-wsl-ubuntu.pin{" "}
```

```
sudo mv cuda-ubuntu2004.pin
```

```
/etc/apt/preferences.d/cuda-repository-pin-600
```

```
wget
```

```
https://developer.download.nvidia.com/compute/cuda/12.2.0/local_installers/cuda-repo-wsl-ubuntu-12-2-local
```

```
sudo dpkg -i cuda-repo-wsl-ubuntu-12-2-local_12.2.0-1_amd64.deb
```

```
sudo cp /var/cuda-repo-wsl-ubuntu-12-2-local/cuda-*keyring.gpg  
/usr/share/keyrings/
```

```
sudo apt-get update
```

```
sudo apt-get -y install cuda
```

7. Set up the required python system installs (Python 3.10).

```
sudo add-apt-repository ppa:deadsnakes/ppa
```

```
sudo apt install python3.10
```

```
sudo apt-get install python3.10-distutils
```

```
curl -sS https://bootstrap.pypa.io/get-pip.py | python3.10
```

8. Create the virtual environment.

```
sudo apt install -y python3.10-venv
```

```
python3 -m venv llmstudio
```

```
source llmstudio/bin/activate
```

9. Clone the H2O LLM Studio repository locally.

```
git clone https://github.com/h2oai/h2o-llmstudio.git
```

```
cd h2o-llmstudio
```

10. Install H2O LLM Studio using the `requirements.txt`.

```
pip install -r requirements.txt
```

11. Run the H2O LLM Studio application.

```
H2O_WAVE_MAX_REQUEST_SIZE=25MB \  
H2O_WAVE_NO_LOG=True \  
H2O_WAVE_PRIVATE_DIR="/download/@output/download" \  
wave run app
```

This will start the H2O Wave server and the H2O LLM Studio app. Navigate to <http://localhost:10101/> (we recommend using Chrome) to access H2O LLM Studio and start fine-tuning your models.

## Install custom package

If required, you can install additional Python packages into your environment. This can be done using pip after activating your virtual environment via `make shell`. For example, to install flash-attention, you would use the following commands:

```
make shell  
pip install flash-attn --no-build-isolation  
pip install git+https://github.com/HazyResearch/flash-attention.git#subdirectory=csrc/rotary
```

Alternatively, you can also directly install the custom package by running the following command.

```
pipenv install package_name
```

## Run H2O LLM Studio

There are several ways to run H2O LLM Studio depending on your requirements.

1. [Run H2O LLM Studio GUI](#)
2. [Run using Docker from a nightly build](#)
3. [Run by building your own Docker image](#)
4. [Run with the CLI \(command-line interface\)](#)

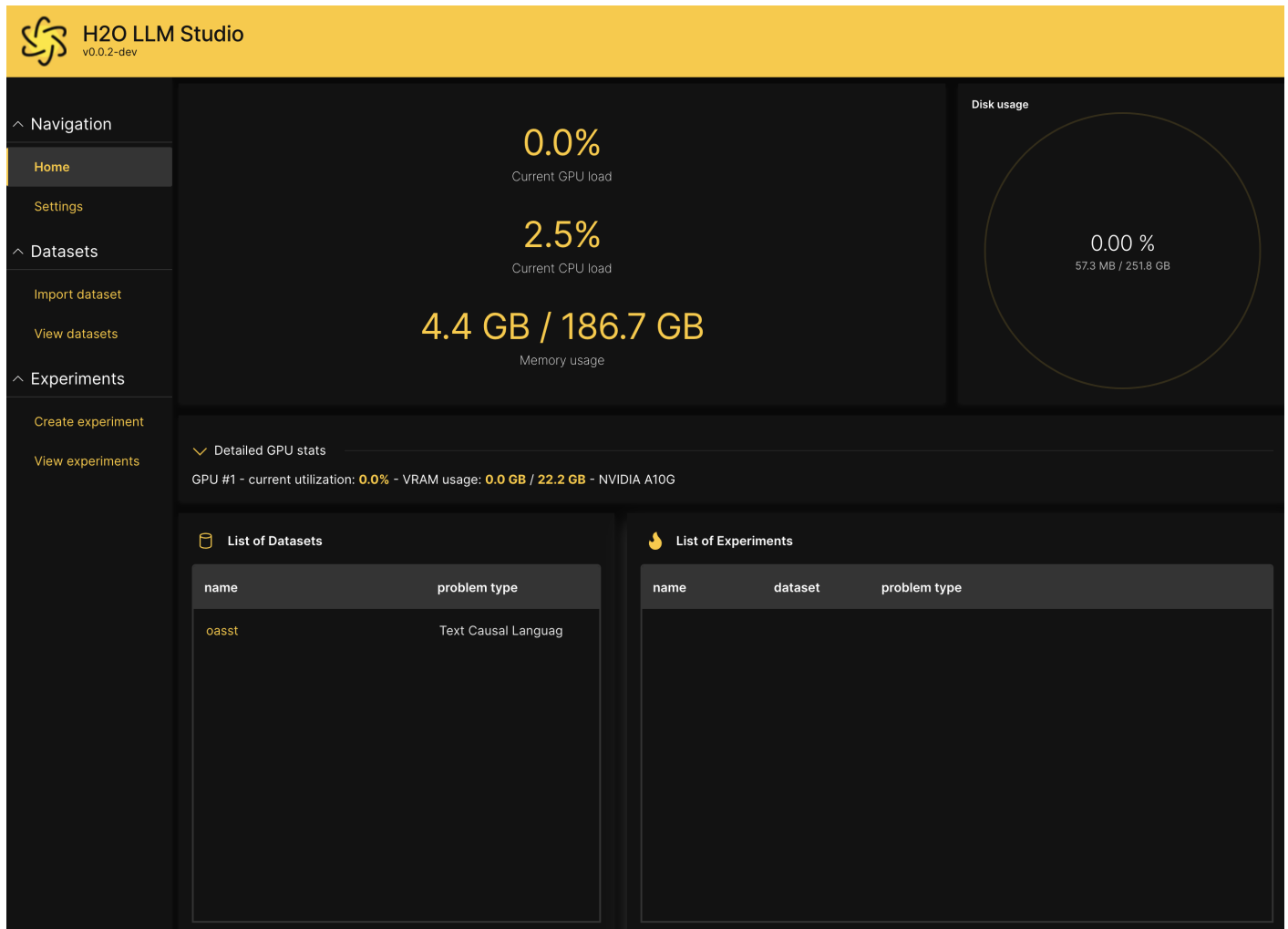
### Run H2O LLM Studio GUI

Run the following command to start the H2O LLM Studio.

```
make llmstudio
```

This will start the H2O Wave server and the H2O LLM Studio app. Navigate to <http://localhost:10101/> (we recommend using Chrome) to access H2O LLM Studio and start fine-tuning your models.





If you are running H2O LLM Studio with a custom environment other than Pipenv, start the app as follows:

```
H2O_WAVE_APP_ADDRESS=http://127.0.0.1:8756 \  
H2O_WAVE_MAX_REQUEST_SIZE=25MB \  
H2O_WAVE_NO_LOG=True \  
H2O_WAVE_PRIVATE_DIR="/download/@output/download" \  
wave run app
```

### Run using Docker from a nightly build

First, install Docker by following the instructions from the [NVIDIA Container Installation Guide](#). H2O LLM Studio images are stored in the `h2oai GCR vorvan` container repository.

```
mkdir -p `pwd`/data  
mkdir -p `pwd`/output  
docker run \  
  --runtime=nvidia \  
  --shm-size=64g \  
  --init \  
  --rm \  
  -p 10101:10101 \  
  -v `pwd`/data:/workspace/data \  
  -v `pwd`/output:/workspace/output \  
  -v ~/.cache:/home/llmstudio/.cache \  
  gcr.io/vorvan/h2oai/h2o-llmstudio:nightly
```

Navigate to <http://localhost:10101/> (we recommend using Chrome) to access H2O LLM Studio and start fine-tuning your models.

- **Note:** Other helpful docker commands are `docker ps` and `docker kill`.

## Run by building your own Docker image

```
docker build -t h2o-llmstudio .
docker run \
  --runtime=nvidia \
  --shm-size=64g \
  --init \
  --rm \
  -p 10101:10101 \
  -v `pwd`/data:/workspace/data \
  -v `pwd`/output:/workspace/output \
  -v ~/.cache:/home/llmstudio/.cache \
  h2o-llmstudio
```

## Run with command line interface (CLI)

You can also use H2O LLM Studio with the command line interface (CLI) and specify the configuration .yaml file that contains all the experiment parameters. To finetune using H2O LLM Studio with CLI, activate the pipenv environment by running `make shell`.

To specify the path to the configuration file that contains the experiment parameters, run:

```
python train.py -Y {path_to_config_yaml_file}
```

To run on multiple GPUs in DDP mode, run:

```
bash distributed_train.sh {NR_OF_GPUS} -Y {path_to_config_yaml_file}
```

- **Note:** By default, the framework will run on the first `k` GPUs. If you want to specify specific GPUs to run on, use the `CUDA_VISIBLE_DEVICES` environment variable before the command.

To start an interactive chat with your trained model, run:

```
python prompt.py -e {experiment_name}
```

`experiment_name` is the output folder of the experiment you want to chat with. The interactive chat will also work with models that were fine-tuned using the GUI.

## H2O LLM Studio performance

Setting up and running H2O LLM Studio requires the following minimal [prerequisites](#). This page lists out the speed and performance metrics of H2O LLM Studio based on different hardware setups.

The following metrics were measured.

- **Hardware setup:** The type and number of computing devices used to train the model.
- **LLM backbone:** The underlying architecture of the language model. For more information, see [LLM backbone](#).
- **Quantization:** A technique used to reduce the size and memory requirements of the model. For more information, see [Quantization](#).
- **Train:** The amount of time it took to train the model in hours and minutes.
- **Validation:** The amount of time it took to validate the mode in hours and minutes.

Hardware setup	LLM backbone	Quantization	Train (hh:mm:ss)	Validation (hh:mm:ss)
8xA10G	h2oai/h2ogpt-4096-llama2-7b	bfloat16	11:35	3:32
4xA10G	h2oai/h2ogpt-4096-llama2-7b	bfloat16	21:13	06:35
2xA10G	h2oai/h2ogpt-4096-llama2-7b	bfloat16	37:04	12:21
1xA10G	h2oai/h2ogpt-4096-llama2-7b	bfloat16	1:25:29	15:50
8xA10G	h2oai/h2ogpt-4096-llama2-7b	nf4	14:26	06:13
4xA10G	h2oai/h2ogpt-4096-llama2-7b	nf4	26:55	11:59
2xA10G	h2oai/h2ogpt-4096-llama2-7b	nf4	48:24	23:37
1xA10G	h2oai/h2ogpt-4096-llama2-7b	nf4	1:26:59	42:17
8xA10G	h2oai/h2ogpt-4096-llama2-13b	bfloat16	OOM	OOM
4xA10G	h2oai/h2ogpt-4096-llama2-13b	bfloat16	OOM	OOM
2xA10G	h2oai/h2ogpt-4096-llama2-13b	bfloat16	OOM	OOM
1xA10G	h2oai/h2ogpt-4096-llama2-13b	bfloat16	OOM	OOM
8xA10G	h2oai/h2ogpt-4096-llama2-13b	nf4	25:07	10:58
4xA10G	h2oai/h2ogpt-4096-llama2-13b	nf4	48:43	21:25
2xA10G	h2oai/h2ogpt-4096-llama2-13b	nf4	1:30:45	42:06
1xA10G	h2oai/h2ogpt-4096-llama2-13b	nf4	2:44:36	1:14:20
8xA10G	h2oai/h2ogpt-4096-llama2-70b	nf4	OOM	OOM
4xA10G	h2oai/h2ogpt-4096-llama2-70b	nf4	OOM	OOM
2xA10G	h2oai/h2ogpt-4096-llama2-70b	nf4	OOM	OOM
1xA10G	h2oai/h2ogpt-4096-llama2-70b	nf4	OOM	OOM
—	—	—	—	—
4xA100 80GB	h2oai/h2ogpt-4096-llama2-7b	bfloat16	7:04	3:55
2xA100 80GB	h2oai/h2ogpt-4096-llama2-7b	bfloat16	13:14	7:23

Hardware setup	LLM backbone	Quantization	Train (hh:mm:ss)	Validation (hh:mm:ss)
1xA100 80GB	h2oai/h2ogpt-4096-llama2-7b	bfloat16	23:36	13:25
4xA100 80GB	h2oai/h2ogpt-4096-llama2-7b	nf4	9:44	6:30
2xA100 80GB	h2oai/h2ogpt-4096-llama2-7b	nf4	18:34	12:16
1xA100 80GB	h2oai/h2ogpt-4096-llama2-7b	nf4	34:06	21:51
4xA100 80GB	h2oai/h2ogpt-4096-llama2-13b	bfloat16	11:46	5:56
2xA100 80GB	h2oai/h2ogpt-4096-llama2-13b	bfloat16	21:54	11:17
1xA100 80GB	h2oai/h2ogpt-4096-llama2-13b	bfloat16	39:10	18:55
4xA100 80GB	h2oai/h2ogpt-4096-llama2-13b	nf4	16:51	10:35
2xA100 80GB	h2oai/h2ogpt-4096-llama2-13b	nf4	32:05	21:00
1xA100 80GB	h2oai/h2ogpt-4096-llama2-13b	nf4	59:11	36:53
4xA100 80GB	h2oai/h2ogpt-4096-llama2-70b	nf4	1:13:33	46:02
2xA100 80GB	h2oai/h2ogpt-4096-llama2-70b	nf4	2:20:44	1:33:42
1xA100 80GB	h2oai/h2ogpt-4096-llama2-70b	nf4	4:23:57	2:44:51

- **Note:** The runtimes were gathered using the default parameters.

Expand to see the default parameters

architecture:

```
backbone_dtype: int4
gradient_checkpointing: true
intermediate_dropout: 0.0
pretrained: true
pretrained_weights: ''
```

augmentation:

```
random_parent_probability: 0.0
skip_parent_probability: 0.0
token_mask_probability: 0.0
```

dataset:

```
add_eos_token_to_answer: true
add_eos_token_to_prompt: true
add_eos_token_to_system: true
answer_column: output
chatbot_author: H2O.ai
chatbot_name: h2oGPT
data_sample: 1.0
data_sample_choice:
- Train
- Validation
limit_chained_samples: false
mask_prompt_labels: true
parent_id_column: None
personalize: false
prompt_column:
- instruction
```

```

system_column: None
text_answer_separator: <|answer|>
text_prompt_start: <|prompt|>
text_system_start: <|system|>
train_dataframe: /data/user/oasst/train_full.pq
validation_dataframe: None
validation_size: 0.01
validation_strategy: automatic
environment:
  compile_model: false
  find_unused_parameters: false
  gpus:
    - '0'
    - '1'
    - '2'
    - '3'
    - '4'
    - '5'
    - '6'
    - '7'
  huggingface_branch: main
  mixed_precision: true
  number_of_workers: 8
  seed: -1
  trust_remote_code: true
  use_fsdp: false
experiment_name: default-8-a10g
llm_backbone: h2oai/h2ogpt-4096-llama2-7b
logging:
  logger: None
  neptune_project: ''
output_directory: /output/...
prediction:
  batch_size_inference: 0
  do_sample: false
  max_length_inference: 256
  metric: BLEU
  metric_gpt_model: gpt-3.5-turbo-0301
  metric_gpt_template: general
  min_length_inference: 2
  num_beams: 1
  num_history: 4
  repetition_penalty: 1.2
  stop_tokens: ''
  temperature: 0.0
  top_k: 0
  top_p: 1.0
problem_type: text_causal_language_modeling
tokenizer:
  add_prompt_answer_tokens: false
  max_length: 512
  padding_quantile: 1.0
training:
  batch_size: 2
  differential_learning_rate: 1.0e-05
  differential_learning_rate_layers: []
  drop_last_batch: true
  epochs: 1
  evaluate_before_training: false

```

```
evaluation_epochs: 1.0
grad_accumulation: 1
gradient_clip: 0.0
learning_rate: 0.0001
lora: true
use_dora: false
lora_alpha: 16
lora_dropout: 0.05
lora_r: 4
lora_target_modules: ''
loss_function: TokenAveragedCrossEntropy
optimizer: AdamW
save_checkpoint: "last"
schedule: Cosine
train_validation_data: false
warmup_epochs: 0.0
weight_decay: 0.0
```

# Model flow

The flow of creating and fine-tuning large language models using H2O LLM Studio can be summarized in the following sequential steps:

- [Step 1: Import a dataset](#)
- [Step 2: Create an experiment](#)
- [Step 3: Monitor an experiment](#)
- [Step 4: Compare experiments](#)
- [Step 5: Export a model to Hugging Face Hub](#)

## Step 1: Import a dataset

As the first step in the experiment flow, prep your data and import your dataset to H2O LLM Studio.

- To learn about supported data connectors and data format, see [Supported data connectors and format](#).
- To learn about how to import a dataset to H2O LLM Studio, see [Import a dataset](#).
- To learn about reviewing and editing a dataset, see [View and manage dataset](#).

## Step 2: Create an experiment

As the second step in the experiment flow, create an experiment using the imported dataset. H2O LLM Studio offers several hyperparameter settings that you can adjust for your experiment model. To ensure that your training process is effective, you may need to specify the [hyperparameters](#) like learning rate, batch size, and the number of epochs. H2O LLM Studio provides an overview of all the parameters you'll need to specify for your experiment.

- To learn about creating a new experiment, see [Create an experiment](#).
- To learn about the settings available for creating an experiment, see [Experiment settings](#).

## Step 3: Monitor an experiment

As the third step in the experiment flow, monitor the launched experiment. H2O LLM Studio allows you to inspect your experiment (model) during and after model training. Simple interactive graphs in H2O LLM Studio allow you to understand the impact of selected hyperparameter values during and after model training. You can then adjust the [hyperparameters](#) to further optimize model performance.

To learn about viewing and monitoring an experiment, see [View and manage experiments](#).

## Step 4: Compare experiments

The H2O LLM studio provides a useful feature that allows comparing various experiments and analyzing how different model parameters affect model performance. This feature is a powerful tool for fine-tuning your machine-learning models and ensuring they meet your desired performance metrics.

To learn about comparing multiple experiments, see [Compare experiments](#).

## Step 5: Export a model to Hugging Face Hub

As the final step in the experiment flow, you can export the fine-tuned model to Hugging Face with a single click.

To learn about exporting a trained model to Hugging Face Hub, see, [Export trained model to Hugging Face](#).

## Core features

### No-code fine-tuning

NLP practitioners can easily fine-tune models without the need for code expertise. The user interface, which is specifically designed for LLMs, allows users to upload large datasets easily and configure [hyperparameters](#) to fine-tune the model.

### Highly customizable (wide range of hyperparameters)

H2O LLM Studio supports a wide variety of hyperparameters that can be used to fine-tune the model and supports the following fine-tuning techniques to enable advanced customization:

- [Low-Rank Adaptation \(LoRA\)](#)
- [8-bit model training with a low memory footprint](#)

### Advanced evaluation metrics and experiment comparison

Advanced evaluation metrics in H2O LLM Studio can be used to validate the answers generated by the LLM. This helps to make data-driven decisions about the model. It also offers visual tracking and comparison of experiment performance, making it easy to analyze and compare different fine-tuned models. You can also visualize how different parameters affect the model performance, and optionally use the [Neptune](#) integration to track and log your experiments.

### Instant publishing models

H2O LLM Studio enables easy model sharing with the community by allowing you to export the model to the [Hugging Face Hub](#) with a single click.

### Instant feedback on model performance

Additionally, H2O LLM Studio lets you chat with the fine-tuned model and receive instant feedback about model performance.



# Concepts

H2O LLM Studio is based on a few key concepts and uses several key terms across its documentation. Each, in turn, is explained within the sections below.

## LLM

A Large Language Model (LLM) is a type of AI model that uses deep learning techniques and uses massive datasets to analyze and generate human-like language. For example, many AI chatbots or AI search engines are powered by LLMs.

Generally speaking, LLMs can be characterized by the following parameters: - size of the training dataset - cost of training (computational power) - size of the model (parameters) - performance after training (or how well the model is able to respond to a particular question)

## Parameters and hyperparameters

In the context of an LLM, parameters and hyperparameters are a crucial part of determining the model's performance and overall behaviour.

- **Parameters:** The internal variables of the model that are learned during the training process. In the case of an LLM, parameters typically include the weights and biases associated with the neural network layers. The values of parameters directly influence the model's predictions and the quality of generated text.
- **Hyperparameters:** The configuration choices that are set before training the model and are not learned directly from the data (e.g., number of epochs, batch size etc.). These choices impact the learning process and influence the model's overall behavior. Hyperparameters need to be tuned and optimized to achieve the best performance. H2O LLM Studio GUI shows tooltips next to each hyperparameter to explain what each hyperparameter is for. You can also see the following references for more details about hyperparameters in H2O LLM Studio.
  - Dataset settings
  - [Experiment settings](#)

## LLM Backbone

LLM Backbone is a key hyperparameter that determines the model's architecture. This option is the most important setting when it comes to experiment creation, as it sets the pretrained model weights. For more information about LLM Backbone, see [Experiment settings](#).

## Generative AI

Generative AI refers to AI models that can generate new content, such as images, videos, or text, that did not exist before. These models learn from large datasets and use this knowledge to create new content that is similar in style or content to the original dataset.

## Foundation model

A particular adaptive model that has been trained on a large amount of data and starts to derive relationships between words and concepts. Foundation models are fine-tuned to become more specific and adapt to the related domain more efficiently.

## Fine-tuning

Fine-tuning refers to the process of taking a pre-trained language model and further training it on a specific task or domain to improve its performance on that task. It is an important technique used to adapt LLMs to specific tasks and domains.

## LoRA (Low-Rank Adaptation)

Low-Rank Adaptation (LoRa) involves modifying the pre-trained model by adjusting its weights and biases to better fit the new task. This adaptation is done in a way that preserves the pre-trained weights from the original dataset while also adjusting for the new task's specific requirements. This method of training or fine-tuning models consumes less memory. By using low rank adaptation, the pre-trained model can be quickly adapted to new tasks, without requiring a large amount of new training data.

## Quantization

Quantization is a technique used to reduce the size and memory requirements of a large language model without sacrificing its accuracy. This is done by converting the floating-point numbers used to represent the model's parameters to lower-precision numbers, such as half-floats or bfloat16. Quantization can be used to make language models more accessible to users with limited computing resources.

### 8-bit model training with a low memory footprint

8-bit model training with a low memory footprint refers to a fine-tuning technique that reduces the memory requirements for training neural networks by using 8-bit integers instead of 32-bit floating-point numbers. This approach can significantly reduce the amount of memory needed to store the model's parameters and can make it possible to train larger models on hardware with limited memory capacity.

## BLEU

Bilingual Evaluation Understudy (BLEU) is a model evaluation metric that is used to measure the quality of the predicted text against the input text.

BLEU: BLEU (Bilingual Evaluation Understudy) measures the quality of machine-generated texts by comparing them to reference texts by calculating a score between 0 and 1, where a higher score indicates a better match with the reference text. BLEU is based on the concept of n-grams, which are contiguous sequences of words. The different variations of BLEU such as BLEU-1, BLEU-2, BLEU-3, and BLEU-4 differ in the size of the n-grams considered for evaluation. BLEU-n measures the precision of n-grams (n consecutive words) in the generated text compared to the reference text. It calculates the precision score by counting the number of overlapping n-grams and dividing it by the total number of n-grams in the generated text.

## Perplexity

Perplexity (PPL) is a commonly used evaluation metric. It measures the confidence a model has in its predictions, or in simpler words how 'perplexed' or surprised it is by seeing new data. Perplexity is defined as the exponentiated cross-entropy of a sequence of tokens. Lower perplexity means the model is highly confident and accurate in the sequence of tokens it is responding with.

# Supported data connectors and format

## Data connectors

H2O LLM Studio supports the following data connectors to access or upload external data sources.

- **Upload:** Upload a local dataset from your machine.
- **Local:** Specify the file location of the dataset on your machine.
- **AWS S3 (Amazon AWS S3):** Connect to an Amazon AWS S3 data bucket.
- **Kaggle:** Connect to a Kaggle dataset.

## Data format

- Each data connector requires either a single `.csv` or `.pq` file, or the data to be in a `.zip` file for a successful import.
- H2O LLM studio requires a `.csv` file with a minimum of two columns, where one contains the instructions and the other has the model's expected output. You can also include an additional validation dataframe in the same format or allow for an automatic train/validation split to assess the model's performance.
- Optionally, a **Parent Id** can be used for training nested data prompts that are linked to a parent question.
- During an experiment you can adapt the data representation with the following settings:
  - **Prompt Column:** The column in the dataset containing the user prompt.
  - **Answer Column:** The column in the dataset containing the expected output.
  - **Parent Id Column:** An optional column specifying the parent id to be used for chained conversations. The value of this column needs to match an additional column with the name `id`. If provided, the prompt will be concatenated after preceding parent rows.
- **Note:** To train a chatbot style model, you need to convert your data into a question and answer format. There are other enterprise solutions by H2O.ai that may help you prep your data. For more information, see [H2O.ai's Generative AI page](#) and this blogpost about [H2O LLM DataStudio: Streamlining Data Curation and Data Preparation for LLMs related tasks](#).

## Example data

H2O LLM Studio provides a sample dataset (converted dataset from [OpenAssistant/oasst2](#)) that can be downloaded [here](#). It is recommended to use `train_full.csv` for training. This dataset is also downloaded and prepared by default when first starting the GUI. Multiple dataframes can be uploaded into a single dataset by uploading a `.zip` archive.

# Import a dataset

H2O LLM Studio provides a number of data connectors to support importing data from local or external sources and requires your data to be in a certain format for successful importing of data.

For more information, see [Supported data connectors and format](#).

## Import data

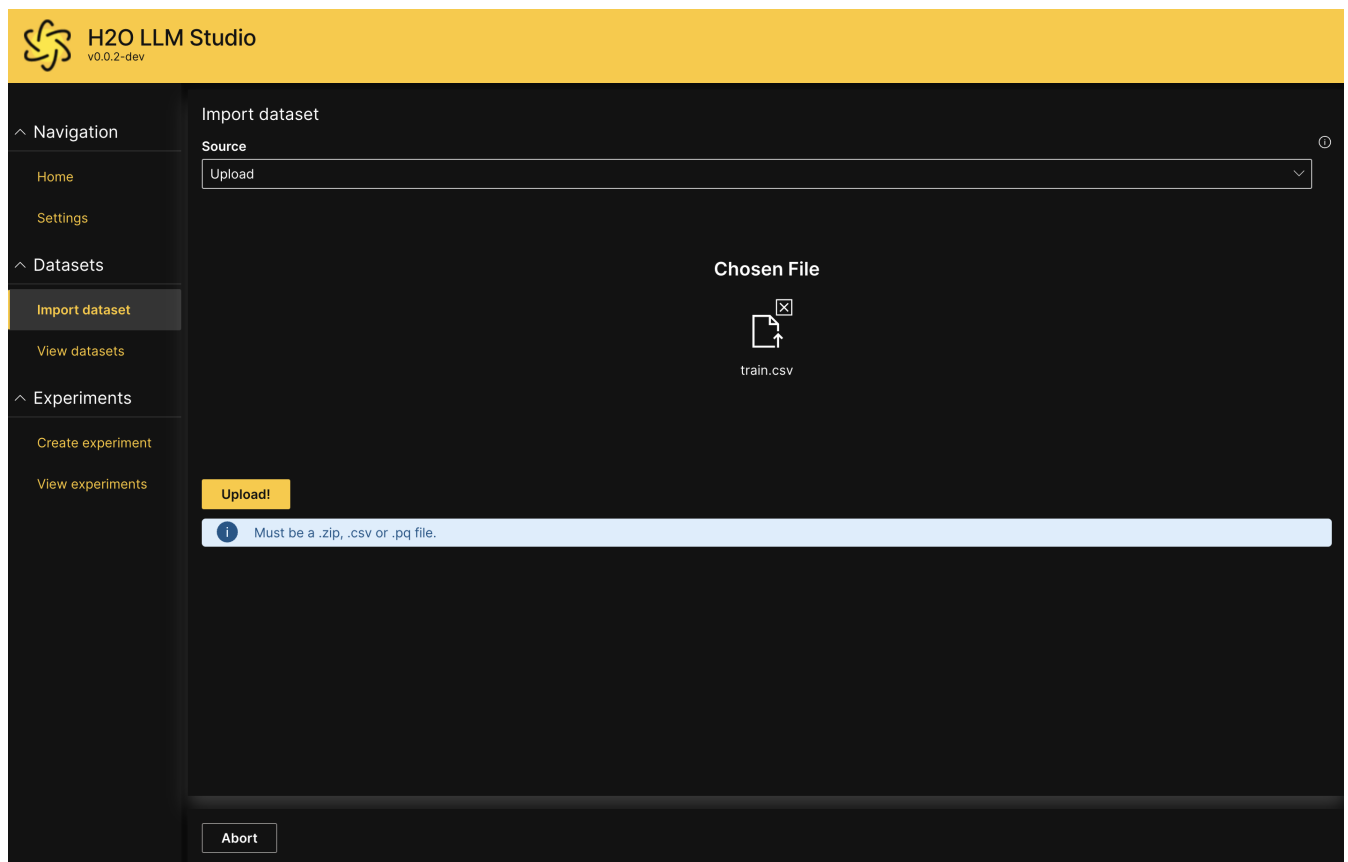
Follow the relevant steps below to import a dataset to H2O LLM Studio.

1. On the H2O LLM Studio left-navigation pane, click **Import dataset**.
2. Select the relevant **Source** (data connector) that you want to use from the dropdown list .

## Data Sources

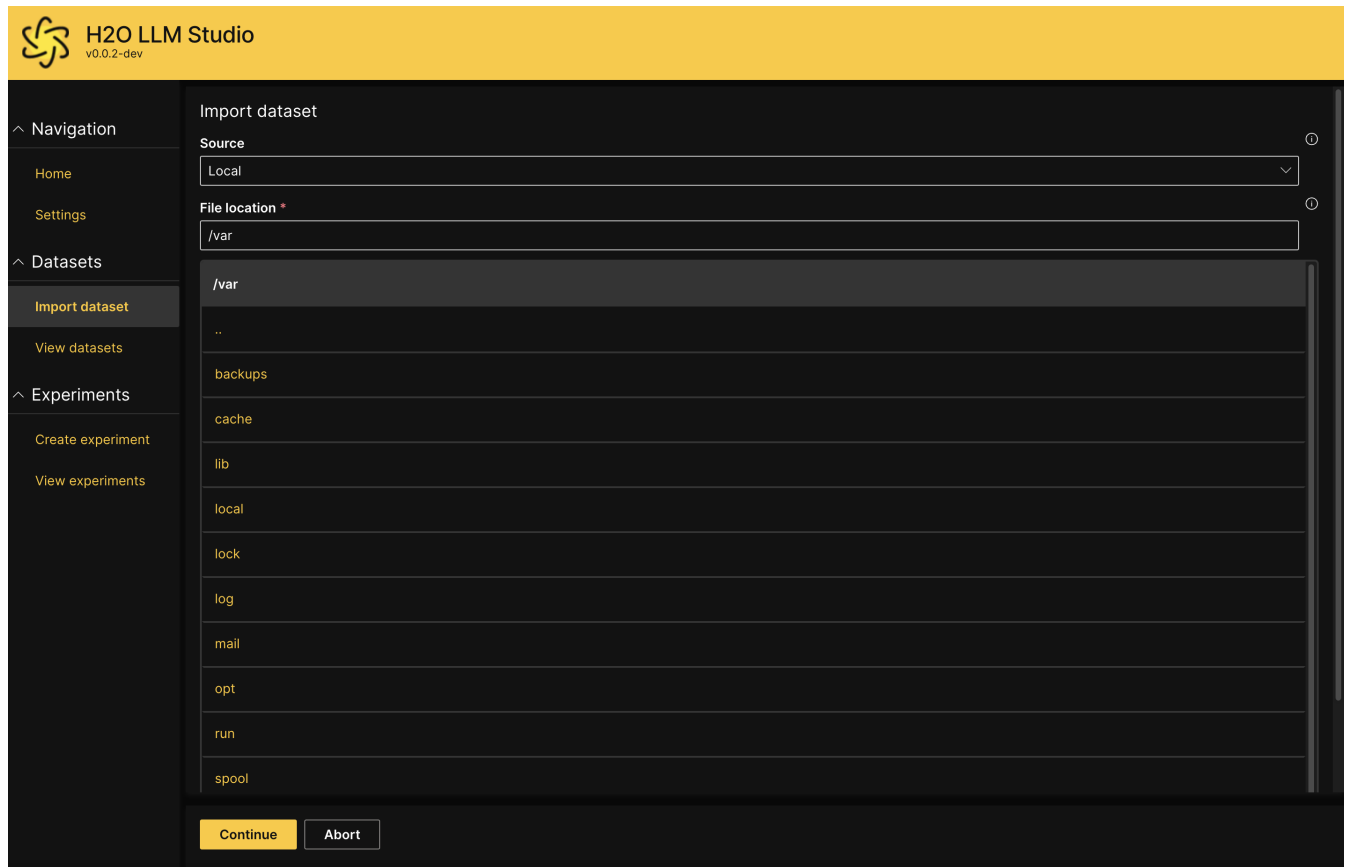
### Upload

1. Drag and drop the file, or click **Browse** and select the file you want to upload.
2. Click **Upload**.



### Local


1. Enter the file path as the **File Location** or select the relevant local directory that the dataset is located in.
2. Click **Continue**.



## AWS S3

1. Enter values for the following fields:

- **S3 bucket name:** The name of the S3 bucket including the relative file paths.
- **AWS access key:** The access key associated with your S3 bucket. This field is optional. If the S3 bucket is public, you can leave this empty for anonymous access.
- **AWS access secret:** The access secret associated with your S3 bucket. This field is optional. If the S3 bucket is public, you can leave this empty for anonymous access.
- **File name:** Enter the file name of the dataset that you want to import.

 H2O LLM Studio  
v0.0.2-dev

^ Navigation

Home

Settings

^ Datasets

Import dataset

View datasets

^ Experiments

Create experiment

View experiments

Import dataset

Source

AWS S3

S3 bucket name \*

bucket\_name

AWS access key \*

.....

AWS secret key \*

.....

File name \*

train.csv

Must be a .zip, .csv or .pq file.

Continue

Abort

For more information, see [AWS credentials](#) and [Methods for accessing a bucket](#) in the AWS Documentation.

2. Click **Continue**.

## Kaggle

1. Enter values for the following fields:

- **Kaggle API command:** Enter the Kaggle API command that you want to execute.
- **Kaggle username:** Your Kaggle username for API authentication
- **Kaggle secret key:** Your Kaggle secret key for API authentication.

2. Click **Continue**.

## Configure dataset

Once you have successfully uploaded or imported your dataset, you can configure the dataset settings. Depending on the problem type, you may need to specify the following:

- **Tip:** You can upload a `.zip` file with both training and validation sets to avoid having to separately upload files.
- **Dataset name:** A suitable name for the whole dataset which includes both the train dataframe and validation dataframe.
- **Problem Type:** Defines the problem type of the experiment, which also defines the settings H2O LLM Studio displays for the experiment.
- **Causal Language Modeling:** Used to fine-tune large language models
- **DPO Modeling:** Used to fine-tune large language models using Direct Preference Optimization
- **Sequence To Sequence Modeling:** Used to fine-tune large sequence to sequence models
- **Causal Classification Modeling:** Used to fine-tune causal classification models
- **Train Dataframe:** Defines a `.csv` or `.pq` file containing a dataframe with training records that H2O LLM Studio uses to *train* the model.
- The records are combined into mini-batches when training the model.
- **Validation Dataframe:** Defines a `.csv` or `.pq` file containing a dataframe with validation records that H2O LLM Studio uses to *evaluate* the model during training.
- The *validation* dataframe should have the same format as the *train* dataframe.
- **System Column:** The column in the dataset containing the system input which is always prepended for a full sample.
- **Prompt Column:** One column or multiple columns in the dataset containing the user prompt. If multiple columns are selected, the columns are concatenated with a separator defined in **Prompt Column Separator**.

- **Rejected Prompt Column:** The column in the dataset containing the user prompt for the rejected answer. By default this can be set to None to take the same prompt as for the accepted answer and should only be changed if the accepted and rejected answers exhibit different prompts, such as when using KTOPairLoss.
- **Answer Column:** The column in the dataset containing the expected output. For classification, this needs to be an integer column starting from zero containing the class label.
- **Rejected Answer Column:** The column in the dataset containing the rejected response, i.e. an answer that is not preferred by the user. See <https://arxiv.org/abs/2305.18290> for more details.
- **Parent Id Column:** An optional column specifying the parent id to be used for chained conversations. The value of this column needs to match an additional column with the name `id`. If provided, the prompt will be concatenated after preceding parent rows.

**H2O LLM Studio**  
v0.0.2

Navigation: Home, Settings, Datasets, View datasets, Experiments (Create experiment, View experiments)

**Configure dataset**

Dataset name \*  
sample-data

Train Dataframe  
questionDoctorQAs.csv

Validation Dataframe  
icliniqQAs.csv

Prompt Column  
question, question\_text

Answer Column  
answer

Parent Id Column  
None

Buttons: Continue, Merge With Existing Dataset, Abort

## Data validity check

H2O LLM Studio will provide a preview of the dataset input (sample questions) and output (sample answers) according to the content of the imported dataset. Review the text to ensure that the input and output is as intended, and then click **Continue**.

## View dataset

You will now be redirected to the **View datasets** screen. You should be able to see the dataset you just imported listed on the screen.



## ^ Navigation

[Home](#)
[Settings](#)

## ^ Datasets

[Import dataset](#)
[View datasets](#)

## ^ Experiments

[Create experiment](#)
[View experiments](#)

name ▾	problem type ▾	train dataframe	train rows	validation rows	labels	
sample-data	Causal Language Modeling	questionDoctorQAs.csv	5679	465	answer	⋮
oasst	Causal Language Modeling	train_full.pq	8274	None	output	⋮

2 of 2

[Import dataset](#)
[Delete datasets](#)

For more information about viewing dataset summary and statistics, see [View and manage datasets](#)

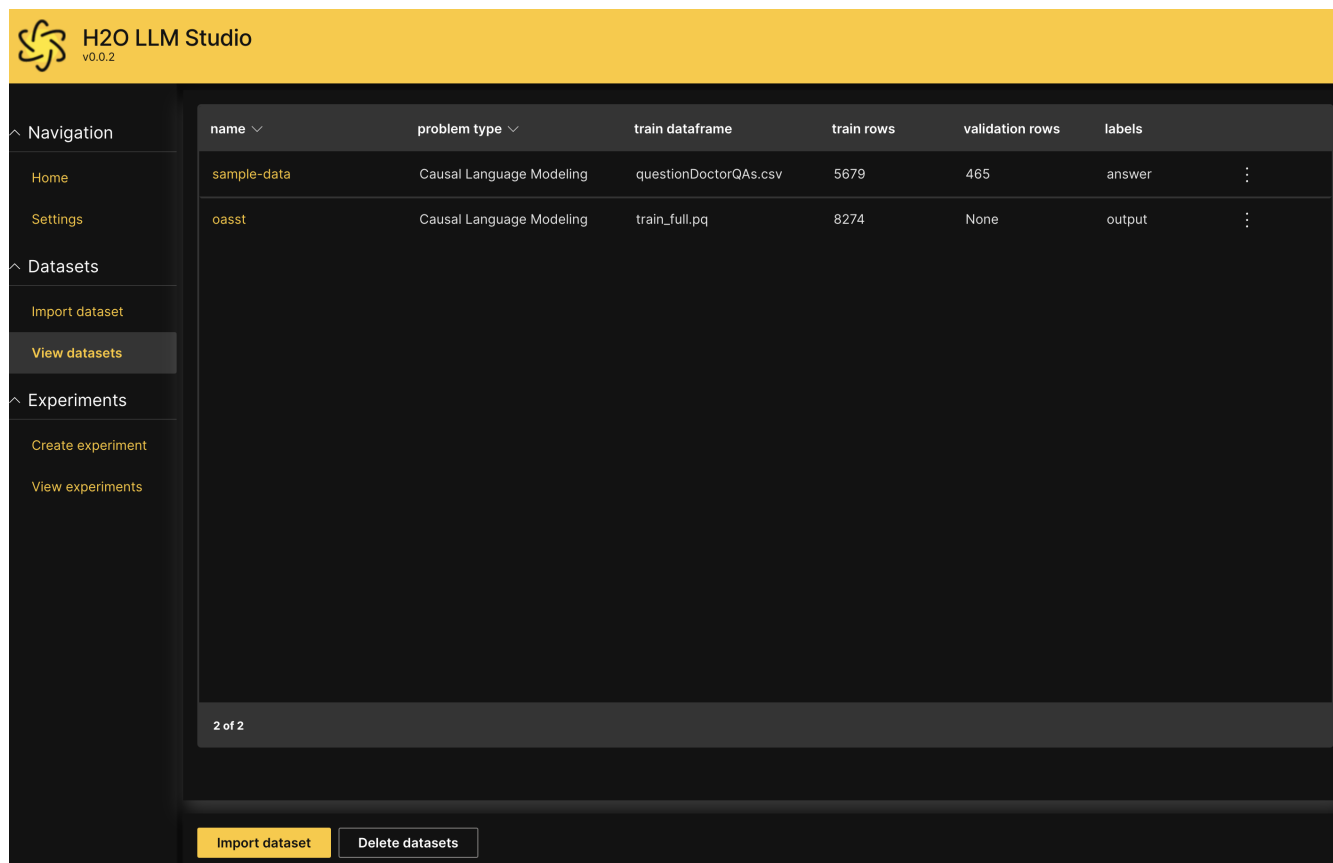
## View and manage dataset

You can view, review, edit, or delete your datasets once you have imported them. You can also start a new experiment using a dataset you have imported.

### View a dataset

To view an imported dataset:

1. On the H2O LLM Studio left-navigation pane, click **View datasets**.
2. You will see the datasets table with a list of all the datasets you have imported so far. Click the name of the dataset that you want to view.



The screenshot shows the H2O LLM Studio interface. The top header is yellow with the H2O logo and 'H2O LLM Studio v0.0.2'. The left sidebar is dark grey with a navigation menu. The main area is dark grey and displays a table of datasets. The table has columns: name, problem type, train dataframe, train rows, validation rows, and labels. Two datasets are listed: 'sample-data' and 'oasst'. Below the table, there is a pagination bar showing '2 of 2'. At the bottom, there are two buttons: 'Import dataset' and 'Delete datasets'.

name	problem type	train dataframe	train rows	validation rows	labels
sample-data	Causal Language Modeling	questionDoctorQAs.csv	5679	465	answer
oasst	Causal Language Modeling	train_full.pq	8274	None	output

- **Note:** For more information about the dataset details you see on the table above, see [dataset configurations](#).

### Dataset tabs

You will see the following tabs that provide details and different aspects of your dataset.

- **Sample train data** : This tab contains sample training data from the imported dataset.
- **Sample train visualization**: This tab visualizes a few sample training data from the imported dataset in a question-answer format; simulating the way the chatbot would answer questions based on the training data.
- **Train data statistics**: This tab contains metrics about the training data (e.g., unique values) from the imported dataset.
- **Summary**: This tab contains the following details about the dataset.

Name	Description
<b>Name</b>	Name of the dataset.
<b>Problem type</b>	Problem type of the dataset.

Name	Description
<b>Train dataframe</b>	Name of the training dataframe in the imported dataset. An imported dataset can contain train, test, and validation dataframes.
<b>Train rows</b>	The number of rows the train dataframe contains.
<b>Validation dataframe</b>	Name of the validation dataframe in the imported dataset. An imported dataset can contain train, test, and validation dataframes.
<b>Validation rows</b>	The number of rows the validation dataframe contains.
<b>Labels</b>	The labels the imported dataset contains.

## Edit a dataset

To edit an imported dataset,

1. On the H2O LLM Studio left-navigation pane, click **View datasets**. You will see the datasets table with a list of all the datasets you have imported so far.
2. Locate the row of the dataset you want to edit and click the more\_vert Kebab menu.
3. Select **Edit dataset**.
4. Make the desired changes to the dataset configuration. You can also [merge the dataset with an existing dataset](#) at this point.
5. Click **Continue** and review the dataset with your changes.

## Delete a dataset

When a dataset is no longer needed, you can delete it. Deleted datasets are permanently removed from the H2O LLM Studio instance.

**Caution:** You can only delete datasets that are not linked to any experiments. If you wish to delete a dataset that is linked to an experiment, first [delete the experiment](#), and then delete the dataset.

1. On the H2O LLM Studio left-navigation pane, click **View datasets**.
2. Click **Delete datasets**.
3. Select the dataset(s) that you want to delete.
4. Click **Delete** to confirm deletion.

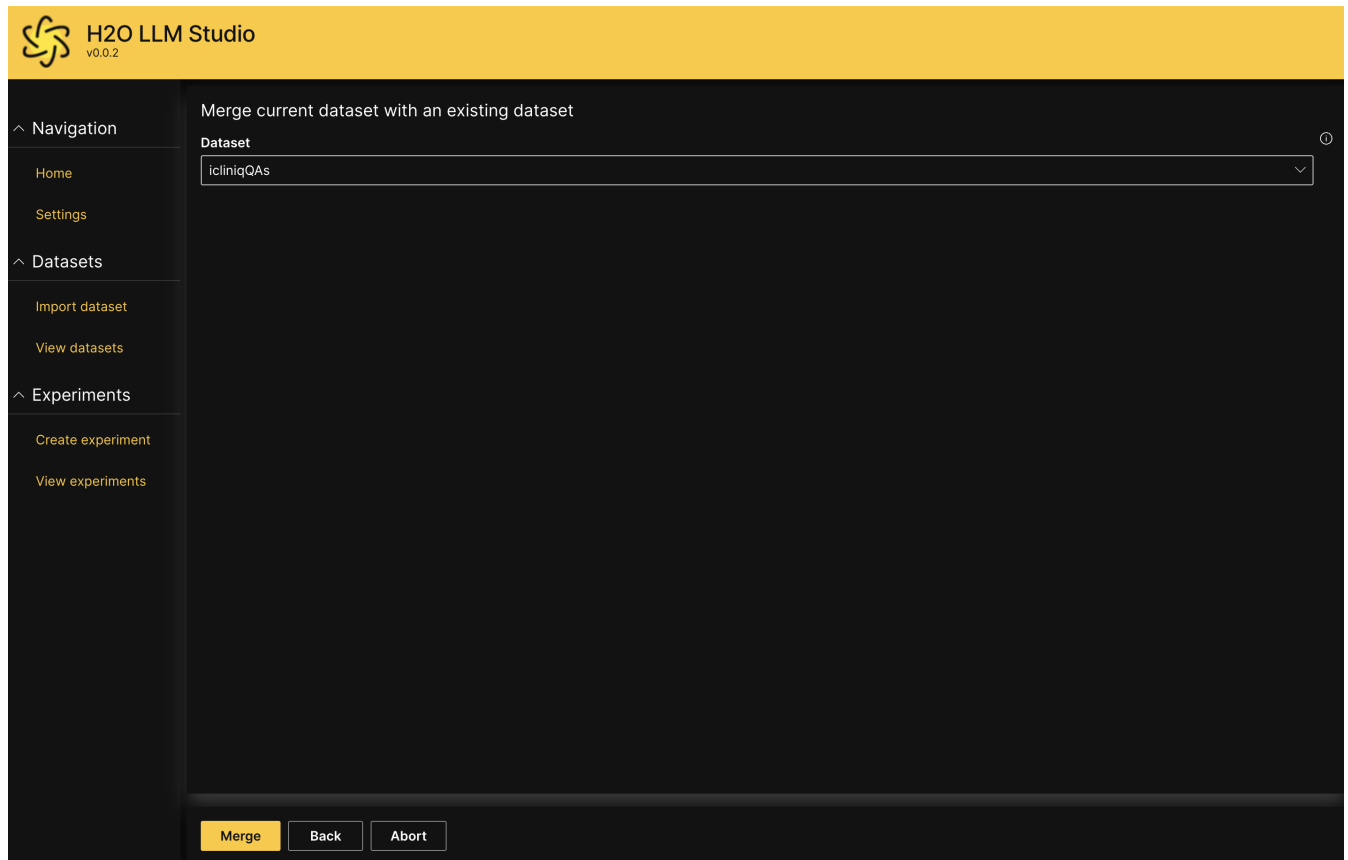
## Merge datasets

H2O LLM Studio enables you to merge imported datasets into one main dataset. This functionality can be used to merge training and validation data together into one dataset or extend your existing dataset with more data and increase your dataset size.

- **Note:** H2O LLM Studio does not merge dataset files in the sense that rows are combined, and duplicate rows are removed. “Merge”, in this case, refers to bringing the dataset files a dataset might have to a single dataset (another dataset), continuing other dataset files already.

Generally, you might want to merge datasets in H2O LLM Studio to have both the training data .csv and validation data .csv in one final dataset.

1. On the H2O LLM Studio left-navigation pane, click **View datasets**.
2. Click the more\_vert Kebab menu of the dataset you want to merge with.
3. Click **Edit dataset**.
4. Click **Merge with existing dataset**.
5. Select the dataset you want that you want to merge with.



6. Click **Merge**.
7. Adjust the dataset configuration if needed. For more information about the configurations, see [Configure dataset](#).
8. Click **Continue**.
9. Review the text to ensure that the input and output is as intended, and then click **Continue**.

Your datasets are now merged.

- **Note:** Alternatively, you can also merge datasets at the point of [importing a dataset](#) or combine both datasets (.csv files) into a .zip file before uploading it as a whole dataset.

# Experiment settings

The settings for creating an experiment are grouped into the following sections:

- [General settings](#)
- [Dataset settings](#)
- [Tokenizer settings](#)
- [Architecture settings](#)
- [Training settings](#)
- [Augmentation settings](#)
- [Prediction settings](#)
- [Environment settings](#)
- [Logging settings](#)

The settings under each category are listed and described below.

## General settings

### Dataset

Select the dataset for the experiment.

### Problem type

Defines the problem type of the experiment, which also defines the settings H2O LLM Studio displays for the experiment.

- Causal Language Modeling: Used to fine-tune large language models
- DPO Modeling: Used to fine-tune large language models using Direct Preference Optimization
- Sequence To Sequence Modeling: Used to fine-tune large sequence to sequence models
- Causal Classification Modeling: Used to fine-tune causal classification models

### Import config from YAML

Defines the `.yaml` file that defines the experiment settings.

- H2O LLM Studio supports a `.yaml` file import and export functionality. You can download the config settings of finished experiments, make changes, and re-upload them when starting a new experiment in any instance of H2O LLM Studio.

### Experiment name

It defines the name of the experiment.

### LLM backbone

The **LLM Backbone** option is the most important setting as it sets the pretrained model weights.

- Use smaller models for quicker experiments and larger models for higher accuracy
- Aim to leverage models pre-trained on tasks similar to your use case when possible
- Select a model from the dropdown list or type in the name of a Hugging Face model of your preference

## Dataset settings

### Train dataframe

Defines a `.csv` or `.pq` file containing a dataframe with training records that H2O LLM Studio uses to *train* the model.

- The records are combined into mini-batches when training the model.

## Validation strategy

Specifies the validation strategy H2O LLM Studio uses for the experiment.

To properly assess the performance of your trained models, it is common practice to evaluate it on separate holdout data that the model has not seen during training. H2O LLM Studio allows you to specify different strategies for this task fitting your needs.

Options - **Custom holdout validation** - Specifies a separate holdout dataframe. - **Automatic holdout validation** - Allows to specify a holdout validation sample size that is automatically generated.

## Validation size

Defines an optional relative size of the holdout validation set. H2O LLM Studio do automatically sample the selected percentage from the full training data, and build a holdout dataset that the model is validated on.

## Data sample

Defines the percentage of the data to use for the experiment. The default percentage is 100% (1).

Changing the default value can significantly increase the training speed. Still, it might lead to a substantially poor accuracy value. Using 100% (1) of the data for final models is highly recommended.

## System column

The column in the dataset containing the system input which is always prepended for a full sample.

## Prompt column

One column or multiple columns in the dataset containing the user prompt. If multiple columns are selected, the columns are concatenated with a separator defined in **Prompt Column Separator**.

## Prompt column separator

If multiple prompt columns are selected, the columns are concatenated with the separator defined here. If only a single prompt column is selected, this setting is ignored.

## Answer column

The column in the dataset containing the expected output.

For classification, this needs to be an integer column starting from zero containing the class label.

## Parent ID column

An optional column specifying the parent id to be used for chained conversations. The value of this column needs to match an additional column with the name `id`. If provided, the prompt will be concatenated after preceding parent rows.

## Text prompt start

Optional text to prepend to each prompt.

## Text answer separator

Optional text to append to each prompt / prepend to each answer.

## Add EOS token to prompt

Adds EOS token at end of prompt.

## Add EOS token to answer

Adds EOS token at end of answer.

## Mask prompt labels

Whether to mask the prompt labels during training and only train on the loss of the answer.

## Tokenizer settings

### Max length

Defines the maximum length of the input sequence H2O LLM Studio uses during model training. In other words, this setting specifies the maximum number of tokens an input text is transformed for model training.

A higher token count leads to higher memory usage that slows down training while increasing the probability of obtaining a higher accuracy value.

In case of Causal Language Modeling, this includes both prompt and answer, or all prompts and answers in case of chained samples.

In Sequence to Sequence Modeling, this refers to the length of the prompt, or the length of a full chained sample.

### Add prompt answer tokens

Adds system, prompt and answer tokens as new tokens to the tokenizer. It is recommended to also set **Force Embedding Gradients** in this case.

### Padding quantile

Defines the padding quantile H2O LLM Studio uses to select the maximum token length per batch. H2O LLM Studio performs padding of shorter sequences up to the specified padding quantile instead of the selected **Max length**. H2O LLM Studio truncates longer sequences.

- Lowering the quantile can significantly increase training runtime and reduce memory usage in unevenly distributed sequence lengths but can hurt performance
- The setting depends on the batch size and should be adjusted accordingly
- No padding is done in inference, and the selected **Max Length** is guaranteed
- Setting to 0 disables padding
- In case of distributed training, the quantile will be calculated across all GPUs

## Architecture settings

### Backbone Dtype

The datatype of the weights in the LLM backbone.

### Gradient Checkpointing

Determines whether H2O LLM Studio activates gradient checkpointing (GC) when training the model. Starting GC reduces the video random access memory (VRAM) footprint at the cost of a longer runtime (an additional forward pass). Turning **On** GC enables it during the training process.

**Caution** Gradient checkpointing is an experimental setting that is not compatible with all backbones or all other settings.

Activating *GC* comes at the cost of a longer training time; for that reason, try training without *GC* first and only activate when experiencing *GPU out-of-memory (OOM)* errors.

### Intermediate dropout

Defines the custom dropout rate H2O LLM Studio uses for intermediate layers in the transformer model.

### Pretrained weights

Allows you to specify a local path to the pretrained weights.

## Training settings

### Loss function

Defines the loss function H2O LLM Studio utilizes during model training. The loss function is a differentiable function measuring the prediction error. The model utilizes gradients of the loss function to update the model weights during training. The options depend on the selected Problem Type.

### Optimizer

Defines the algorithm or method (optimizer) to use for model training. The selected algorithm or method defines how the model should change the attributes of the neural network, such as weights and learning rate. Optimizers solve optimization problems and make more accurate updates to attributes to reduce learning losses.

Options:

- **Adadelta**
  - To learn about Adadelta, see ADADELTA: An Adaptive Learning Rate Method.
- **Adam**
  - To learn about Adam, see Adam: A Method for Stochastic Optimization.
- **AdamW**
  - To learn about AdamW, see Decoupled Weight Decay Regularization.
- **AdamW8bit**
  - To learn about AdamW, see Decoupled Weight Decay Regularization.
- **RMSprop**
  - To learn about RMSprop, see Neural Networks for Machine Learning.
- **SGD**
  - H2O LLM Studio uses a stochastic gradient descent optimizer.

### Learning rate

Defines the learning rate H2O LLM Studio uses when training the model, specifically when updating the neural network's weights. The learning rate is the speed at which the model updates its weights after processing each mini-batch of data.

- Learning rate is an important setting to tune as it balances under- and overfitting.
- The number of epochs highly impacts the optimal value of the learning rate.

### Differential learning rate layers

Defines the learning rate to apply to certain layers of a model. H2O LLM Studio applies the regular learning rate to layers without a specified learning rate.

- **Backbone**
  - H2O LLM Studio applies a different learning rate to a body of the neural network architecture.
- **Value Head**
  - H2O LLM Studio applies a different learning rate to a value head of the neural network architecture.

A common strategy is to apply a lower learning rate to the backbone of a model for better convergence and training stability.

### Freeze layers

An optional list of layers to freeze during training. Full layer names will be matched against selected substrings. Only available without LoRA training.

### Use Flash Attention 2

If enabled, Flash Attention 2 will be used to compute the attention. Otherwise, the attention will be computed using the standard attention mechanism.

Flash Attention 2 is a new attention mechanism that is faster and more memory efficient than the standard attention mechanism. Only newer GPUs support this feature.

See <https://arxiv.org/abs/2205.14135> for more details.



## Batch size

Defines the number of training examples a mini-batch uses during an iteration of the training model to estimate the error gradient before updating the model weights. **Batch size** defines the batch size used per a single GPU.

During model training, the training data is packed into mini-batches of a fixed size.

## Epochs

Defines the number of epochs to train the model. In other words, it specifies the number of times the learning algorithm goes through the entire training dataset.

- The **Epochs** setting is an important setting to tune because it balances under- and overfitting.
- The learning rate highly impacts the optimal value of the epochs.
- H2O LLM Studio enables you to utilize a pre-trained model trained on zero epochs (where H2O LLM Studio does not train the model and the pretrained model (experiment) can be evaluated as-is):

## Schedule

Defines the learning rate schedule H2O LLM Studio utilizes during model training. Specifying a learning rate schedule prevents the learning rate from staying the same. Instead, a learning rate schedule causes the learning rate to change over iterations, typically decreasing the learning rate to achieve a better model performance and training convergence.

Options - **Constant** - H2O LLM Studio applies a constant learning rate during the training process. - **Cosine** - H2O LLM Studio applies a cosine learning rate that follows the values of the cosine function. - **Linear** - H2O LLM Studio applies a linear learning rate that decreases the learning rate linearly.

## Warmup epochs

Defines the number of epochs to warm up the learning rate where the learning rate should increase linearly from 0 to the desired learning rate. Can be a fraction of an epoch.

## Weight decay

Defines the weight decay that H2O LLM Studio uses for the optimizer during model training.

Weight decay is a regularization technique that adds an L2 norm of all model weights to the loss function while increasing the probability of improving the model generalization.

## Gradient clip

Defines the maximum norm of the gradients H2O LLM Studio specifies during model training. Defaults to **0**, no clipping. When a value greater than 0 is specified, H2O LLM Studio modifies the gradients during model training. H2O LLM Studio uses the specified value as an upper limit for the norm of the gradients, calculated using the Euclidean norm over all gradients per batch.

This setting can help model convergence when extreme gradient values cause high volatility of weight updates.

## Grad accumulation

Defines the number of gradient accumulations before H2O LLM Studio updates the neural network weights during model training.

- Grad accumulation can be beneficial if only small batches are selected for training. With gradient accumulation, the loss and gradients are calculated after each batch, but it waits for the selected accumulations before updating the model weights. You can control the batch size through the **Batch size** setting.
- Changing the default value of *Grad Accumulation* might require adjusting the learning rate and batch size.

## Lora

Whether to use low rank approximations (LoRA) during training.

## Use Dora

Enables Weight-Decomposed Low-Rank Adaptation (DoRA) to be used instead of low rank approximations (LoRA) during training. This parameter efficient training method is built on top of LoRA and has shown promising results. Especially at lower ranks (e.g.  $r=4$ ), it is expected to perform superior to LoRA.

## Lora R

The dimension of the matrix decomposition used in LoRA.

## Lora Alpha

The scaling factor for the lora weights.

## Lora dropout

The probability of applying dropout to the LoRA weights during training.

## Lora target modules

The modules in the model to apply the LoRA approximation to. Defaults to all linear layers.

## Lora unfreeze layers

An optional list of backbone layers to unfreeze during training. By default, all backbone layers are frozen when training with LoRA, here certain layers can be additionally trained, such as embedding or head layer. Full layer names will be matched against selected substrings. Only available with LoRA training.

## Save checkpoint

Specifies how H2O LLM Studio should save the model checkpoints.

When set to **Last** it will always save the last checkpoint, this is the recommended setting.

When set to **Best** it saves the model weights for the epoch exhibiting the best validation metric. - This setting should be turned on with care as it has the potential to lead to overfitting of the validation data. - The default goal should be to attempt to tune models so that the last epoch is the best epoch.

- Suppose an evident decline for later epochs is observed in logging. In that case, it is usually better to adjust hyperparameters, such as reducing the number of epochs or increasing regularization, instead of turning this setting on.

When set to **Each evaluation epoch** it will save the model weights for each evaluation epoch. - This can be useful for debugging and experimenting, but will consume more disk space. - Models uploaded to Hugging Face Hub will only contain the last checkpoint. - Local downloads will contain all checkpoints.

When set to **Disable** it will not save the checkpoint at all. This can be useful for debugging and experimenting in order to save disk space, but will disable certain functionalities like chatting or pushing to HF.

## Evaluation epochs

Defines the number of epochs H2O LLM Studio uses before each validation loop for model training. In other words, it determines the frequency (in a number of epochs) to run the model evaluation on the validation data.

- Increasing the number of *Evaluation Epochs* can speed up an experiment.
- The **Evaluation epochs** setting is available only if the following setting is turned **Off**: **Save Best Checkpoint**.
- Can be a fraction of an epoch

## Evaluate before training

This option lets you evaluate the model before training, which can help you judge the quality of the LLM backbone before fine-tuning.

## Train validation data

Defines whether the model should use the entire train and validation dataset during model training. When turned **On**, H2O LLM Studio uses the whole train dataset and validation data to train the model.

- H2O LLM Studio also evaluates the model on the provided validation fold. Validation is always only on the provided validation fold.
- H2O LLM Studio uses both datasets for model training if you provide a train and validation dataset.
  - To define a training dataset, use the **Train dataframe** setting.
  - To define a validation dataset, use the **Validation dataframe** setting.
- Turning **On** the **Train validation data** setting should produce a model that you can expect to perform better because H2O LLM Studio trained the model on more data. Though, also note that using the entire train dataset and validation dataset generally causes the model's accuracy to be *overstated* as information from the validation data is incorporated into the model during the training process.

## Augmentation settings

### Token mask probability

Defines the random probability of the input text tokens to be randomly masked during training.

- Increasing this setting can be helpful to avoid overfitting and apply regularization
- Each token is randomly replaced by a masking token based on the specified probability

### Skip parent probability

If **Parent Column** is set, this random augmentation will skip parent concatenation during training at each parent with this specified probability.

### Random parent probability

While training, each sample will be concatenated to a random other sample simulating unrelated chained conversations. Can be specified without using a **Parent Column**.

### Neftune noise alpha

Will add noise to the input embeddings as proposed by <https://arxiv.org/abs/2310.05914> (NEFTune: Noisy Embeddings Improve Instruction Finetuning)

## Prediction settings

### Metric

Defines the metric to evaluate the model's performance.

We provide several metric options for evaluating the performance of your model. The options depend on the selected Problem Type:

Causal Language Modeling, DPO Modeling, Sequence to Sequence Modeling - In addition to the BLEU and the Perplexity score, we offer GPT metrics that utilize the OpenAI API to determine whether the predicted answer is more favorable than the ground truth answer. - To use these metrics, you can either export your OpenAI API key as an environment variable before starting LLM Studio, or you can specify it in the Settings Menu within the UI.

Causal Classification Modeling - AUC: Compute Area Under the Receiver Operating Characteristic Curve (ROC AUC). - Accuracy: Compute the accuracy of the model. - LogLoss: Compute the log loss of the model.

### Metric GPT model

Defines the OpenAI model endpoint for the GPT metric.

### Metric GPT template

The template to use for GPT-based evaluation. Note that for mt-bench, the validation dataset will be replaced accordingly; to approximate the original implementation as close as possible, we suggest to use gpt-4-0613 as the gpt judge model and use 1024 for the max length inference.

### Min length inference

Defines the min length value H2O LLM Studio uses for the generated text.

- This setting impacts the evaluation metrics and should depend on the dataset and average output sequence length that is expected to be predicted.

### Max length inference

Defines the max length value H2O LLM Studio uses for the generated text.

- Similar to the **Max Length** setting in the *tokenizer settings* section, this setting specifies the maximum number of tokens to predict for a given prediction sample.
- This setting impacts the evaluation metrics and should depend on the dataset and average output sequence length that is expected to be predicted.

### Batch size inference

Defines the size of a mini-batch uses during an iteration of the inference. **Batch size** defines the batch size used per GPU.

### Do sample

Determines whether to sample from the next token distribution instead of choosing the token with the highest probability. If turned **On**, the next token in a predicted sequence is sampled based on the probabilities. If turned **Off**, the highest probability is always chosen.

### Num beams

Defines the number of beams to use for beam search. *Num Beams* default value is 1 (a single beam); no beam search.

A higher *Num Beams* value can increase prediction runtime while potentially improving accuracy.

### Temperature

Defines the temperature to use for sampling from the next token distribution during validation and inference. In other words, the defined temperature controls the randomness of predictions by scaling the logits before applying softmax. A higher temperature makes the distribution more random.

- Modify the temperature value if you have the **Do Sample** setting enabled (**On**).
- To learn more about this setting, refer to the following article: [How to generate text: using different decoding methods for language generation with Transformers](https://arxiv.org/pdf/1909.05858.pdf).

### Repetition penalty

The parameter for repetition penalty. 1.0 means no penalty. See <https://arxiv.org/pdf/1909.05858.pdf> for more details.

### Stop tokens

Will stop generation at occurrence of these additional tokens; multiple tokens should be split by comma ,.

### Top K

If  $> 0$ , only keep the top k tokens with the highest probability (top-k filtering).

### Top P

If  $< 1.0$ , only keep the top tokens with cumulative probability  $\geq \text{top\_p}$  (nucleus filtering).

## Environment settings

### GPUs

Determines the list of GPUs H2O LLM Studio can use for the experiment. GPUs are listed by name, referring to their system ID (starting from 1).

## Mixed precision

Determines whether to use mixed-precision. When turned **Off**, H2O LLM Studio does not use mixed-precision.

Mixed-precision is a technique that helps decrease memory consumption and increases training speed.

## Compile model

Compiles the model with Torch. Experimental!

## Find unused parameters

In Distributed Data Parallel (DDP) mode, `prepare_for_backward()` is called at the end of DDP forward pass. It traverses the autograd graph to find unused parameters when `find_unused_parameters` is set to `True` in DDP constructor.

Note that traversing the autograd graph introduces extra overheads, so applications should only set to `True` when necessary.

## Trust remote code

Trust remote code. This can be necessary for some models that use code which is not (yet) part of the `transformers` package. Should always be checked with this option being switched **Off** first.

## Huggingface branch

The **Huggingface Branch** defines which branch to use in a Huggingface repository. The default value is “main”.

## Number of workers

Defines the number of workers H2O LLM Studio uses for the *DataLoader*. In other words, it defines the number of CPU processes to use when reading and loading data to GPUs during model training.

## Seed

Defines the random seed value that H2O LLM Studio uses during model training. It defaults to -1, an arbitrary value. When the value is modified (not -1), the random seed allows results to be reproducible—defining a seed aids in obtaining predictable and repeatable results every time. Otherwise, not modifying the default seed value (-1) leads to random numbers at every invocation.

## Logging settings

### Logger

Defines the logger type that H2O LLM Studio uses for model training

Options - **None** - H2O LLM Studio does not use any logger. - **Neptune** - H2O LLM Studio uses Neptune as a logger to track the experiment. To use Neptune, you must specify a **Neptune API token** and a **Neptune project**.

### Neptune project

Defines the Neptune project to access if you selected Neptune in the **Logger** setting.

# Create an experiment

Follow the relevant steps below to create an experiment in H2O LLM Studio.

1. On the H2O LLM Studio left-navigation pane, click **Create experiment**. Alternatively, you can click **New experiment** on the more\_vert Kebab menu of the [View datasets](#) page.
2. Select the **Dataset** you want to use to fine-tune an LLM model.
3. Select the **Problem type**.
4. Provide a meaningful **Experiment name**.
5. Define the parameters. The most important parameters are:
  - **LLM Backbone**: This parameter determines the LLM architecture to use. It is the foundation model that you continue training. H2O LLM Studio has a predefined list of recommended foundation models available in the dropdown list. You can also type in the name of a [Hugging Face model](#) that is not in the list, for example: `h2oai/h2o-danube2-1.8b-sft` or the path of a local folder that has the model you would like to fine-tune.
  - **Mask Prompt Labels**: This option controls whether to mask the prompt labels during training and only train on the loss of the answer.
  - Hyperparameters such as **Learning rate**, **Batch size**, and number of epochs determine the training process. You can refer to the tooltips that are shown next to each hyperparameter in the GUI to learn more about them.
  - **Evaluate Before Training**: This option lets you evaluate the model before training, which can help you judge the quality of the LLM backbone before fine-tuning.

H2O LLM Studio provides several metric options for evaluating the performance of your model. In addition to the BLEU score, H2O LLM Studio also offers the GPT3.5 and GPT4 metrics that utilize the OpenAI API to determine whether the predicted answer is more favorable than the ground truth answer. To use these metrics, you can either export your OpenAI API key as an environment variable before starting LLM Studio, or you can specify it in the **Settings** menu within the UI.

- **Note:** H2O LLM Studio provides an overview of all the parameters you need to specify for your experiment. The default settings are suitable when you first start an experiment. To learn more about the parameters, see [Experiment settings](#).

6. Click **Run experiment**.

The screenshot shows the 'Create experiment' interface in H2O LLM Studio. On the left is a navigation sidebar with sections: 'Navigation' (Home, Settings), 'Datasets' (Import dataset, View datasets), and 'Experiments' (Create experiment, View experiments). The 'Create experiment' option is selected. The main panel is titled 'General settings' and contains the following fields:

- Dataset \***: A dropdown menu showing 'oasst'.
- Problem Type \***: A dropdown menu showing 'Causal Language Modeling'.
- Import config from YAML**: A toggle switch set to 'Off'.
- Experiment Name**: A text input field containing 'test-experiment'.
- LLM Backbone**: A dropdown menu showing 'EleutherAI/pythia-2.8b-deduped'.

Below the 'General settings' is a section titled 'Dataset settings' with the following fields:

- Train Dataframe**: A dropdown menu showing 'train\_full.pq'.
- Validation Strategy**: A dropdown menu showing 'Automatic holdout validation'.
- Validation Size**: A slider control set to '0.01'.
- Data Sample**: A slider control set to '1'.
- Prompt Column**: A text input field.

At the bottom of the main panel is a yellow button labeled 'Run experiment'.

## Run an experiment on the OASST data via CLI

The steps below provide an example of how to run an experiment on [OASST](#) data via the command line interface (CLI).

1. Get the training dataset (`train_full.csv`), [OpenAssistant Conversations Dataset OASST2](#) and place it into the `examples/data_oasst2` folder; or download it directly using the [Kaggle API](#) command given below.

```
kaggle kernels output philippsinger/openassistant-conversations-dataset-oasst2 -p examples/data_oasst2/
```

2. Go into the interactive shell or open a new terminal window. Install the dependencies first, if you have not installed them already.

```
make setup # installs all dependencies
make shell
```

3. Run the following command to run the experiment.

```
python train.py -Y examples/example_oasst2.yaml
```

After the experiment is completed, you can find all output artifacts in the `examples/output_oasst2` folder. You can then use the `prompt.py` script to chat with your model.

```
python prompt.py -e examples/output_oasst2
```

## View and manage experiments

You can view, rename, stop, or delete your experiments once you launch them. You can also create a new experiment based on an experiment you have already launched.

### View an experiment

To view an experiment:

1. On the H2O LLM Studio left-navigation pane, click **View experiments**.
2. You will see the experiments table with a list of all the experiments you have launched so far. Click the name of the experiment that you want to view.

**H2O LLM Studio** v0.0.2

Navigation: Home, Settings, Datasets (Import dataset, View datasets), Experiments (Create experiment, View experiments)

Search:

name	dataset	metric	val metric	progress	status	info
Test-experiment	oasst	BLEU	4.4217	100%	finished	Runtime: 00:27:16
H2O-LLM	oasst	BLEU	4.5922	100%	finished	Runtime: 00:27:27

2 of 2

Buttons: Refresh, Compare experiments, Stop experiments, Delete experiments

### Experiment tabs

Once you click the name of the experiment, you will see the following tabs that provide details and different aspects of your experiment.

- **Charts** : This tab visually represents the train/validation loss, metrics, and learning rate. These charts allow you to easily track your model's performance as it trains.

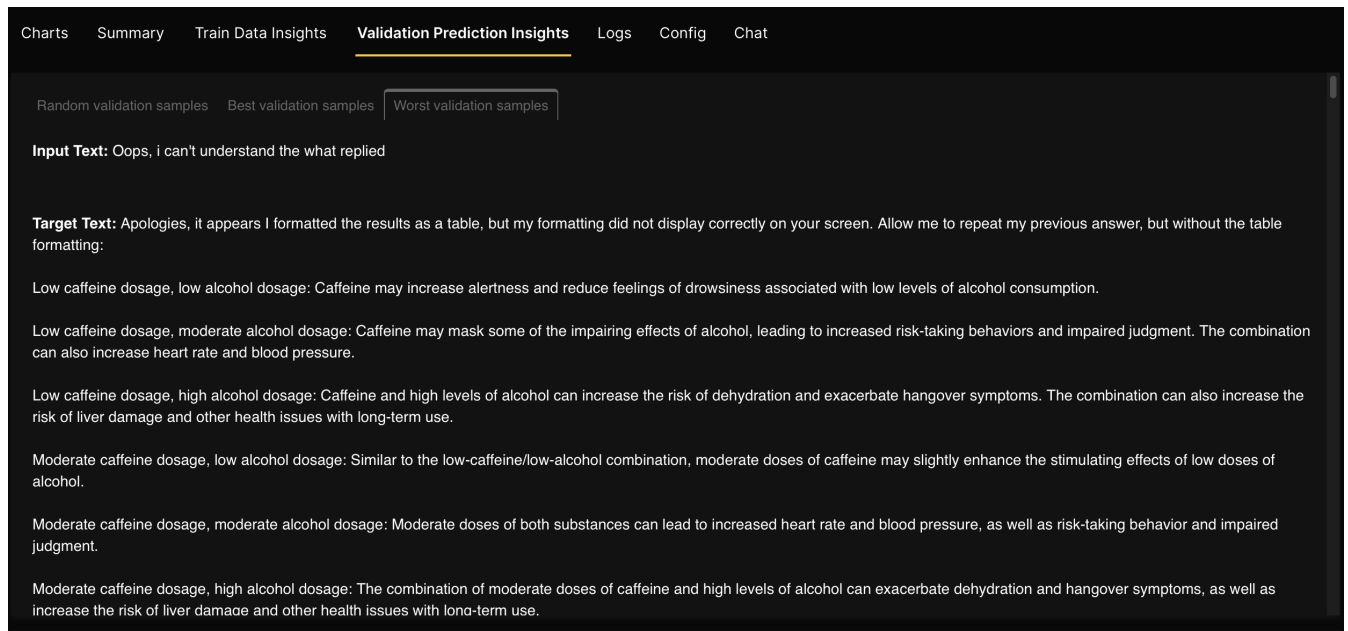
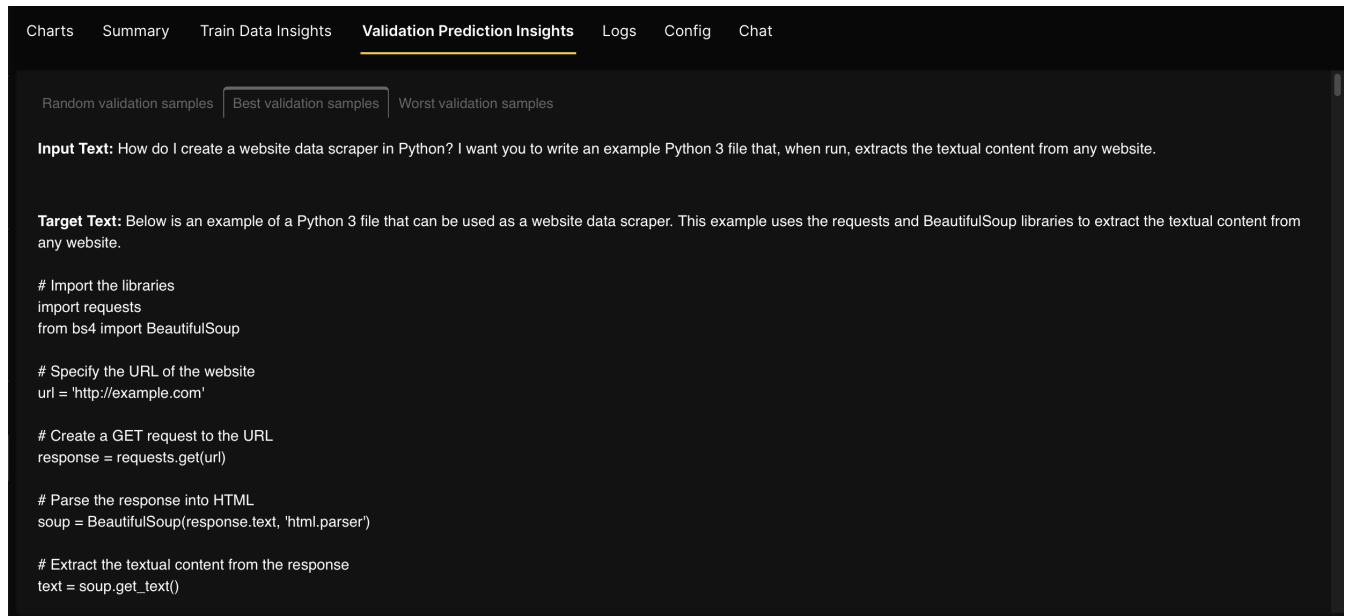




- **Summary** : This tab contains the following details about an experiment.

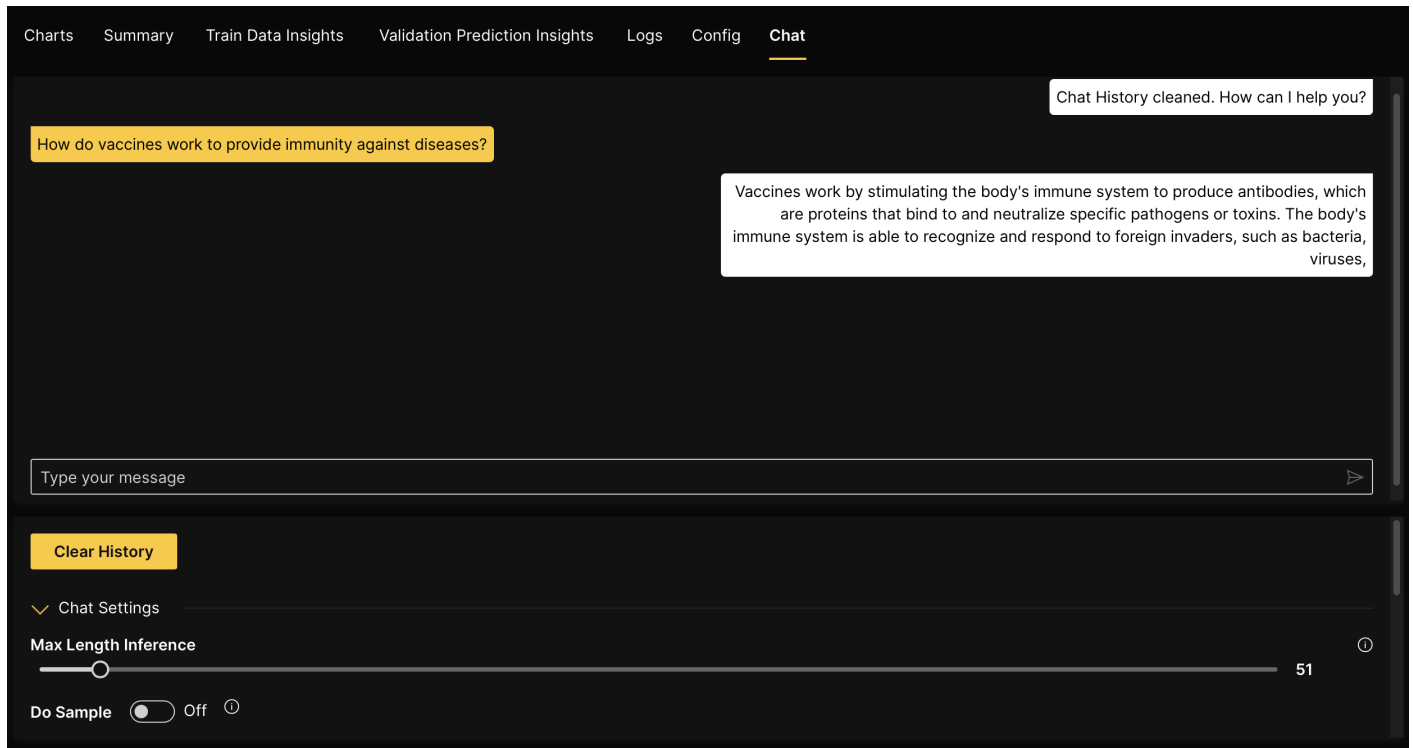
Name	Description
<b>Name</b>	Name of the experiment.
<b>Dataset</b>	Name of the dataset.
<b>Problem type</b>	The problem type of the experiment.
<b>Seed</b>	The random seed value that H2O LLM Studio uses during model training.
<b>GPU list</b>	The list of GPUs H2O LLM Studio can use for the experiment.
<b>Loss</b>	The loss function.
<b>Metric</b>	The metric to evaluate the model's performance.
<b>Val metric</b>	The measure of how well the experiment was performed.

- **Train data insights** : This tab displays the model's first batch, so you can verify that the input data representation is correct. Also, it provides insight into how your data is being processed and can help identify potential issues early on in the experiment.
- **Validation prediction insights** : This tab displays model predictions for random, best, and worst validation samples. This tab becomes available after the first validation run and allows you to evaluate how well your model generalizes to new data.



The **Worst validation samples** give you an idea of where the model is having issues, and the model can be used to fine-tune further.

- **Logs and Config tabs** : These two tabs show you the logs and configuration of the experiment. You can keep track of any changes made and quickly troubleshoot the issues that arise.
- **Chat** : This tab provides a unique opportunity to interact with your trained model and get instant feedback on its performance. The **Chat** tab becomes available after the training is completed and can be used to evaluate how well your model performs in a conversational setting.
- **Note:** You can use the **Chat** feature only when there are no other experiments running. The chatbot is unavailable if the GPU is occupied by another experiment.

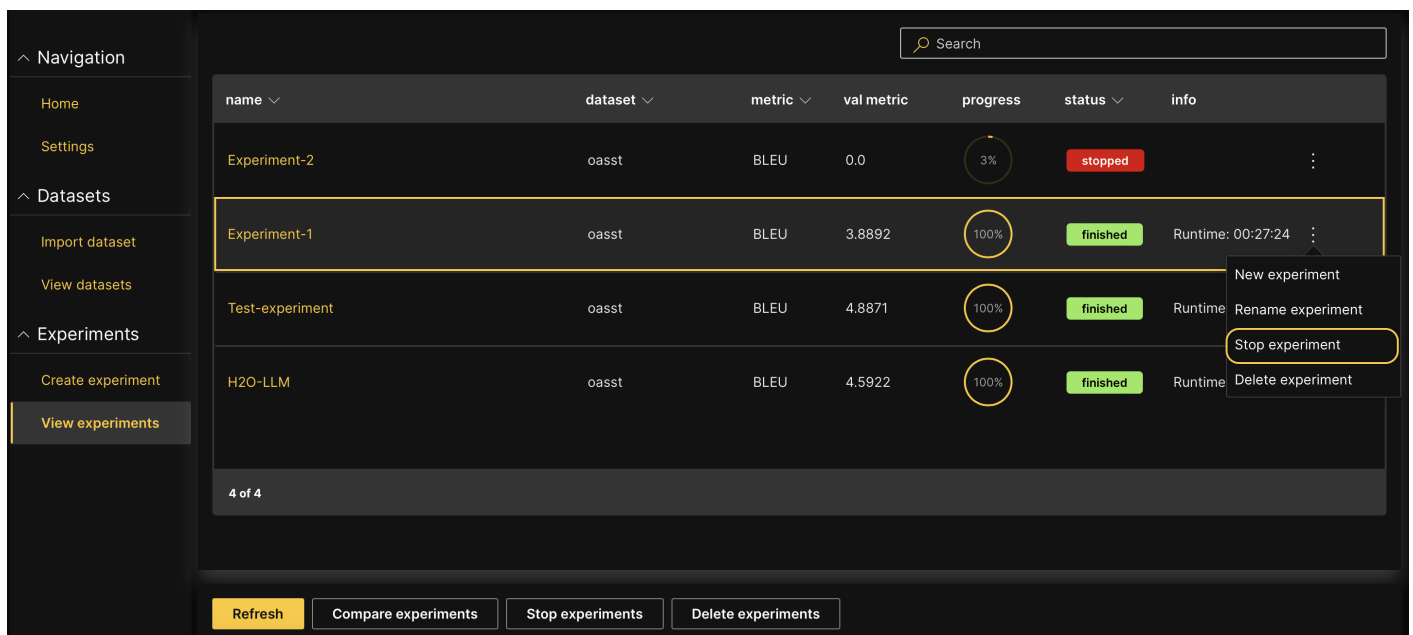


## Stop an experiment

You can stop a running experiment if you no longer need it to be completed.

1. On the H2O LLM Studio left-navigation pane, click **View experiments**.
2. Click **Stop experiments**.
3. Select the experiment(s) that you want to stop.
4. Click **Stop experiments**.

You can also click **Stop experiment** on the more\_vert Kebab menu of the relevant experiment row to stop an experiment from running.



## Delete an experiment

When an experiment is no longer needed, you can delete it. Deleted experiments are permanently removed from the H2O LLM Studio instance.

1. On the H2O LLM Studio left-navigation pane, click **View experiments**.
2. Click **Delete experiments**.
3. Select the experiment(s) that you want to delete and click **Delete experiments**.
4. Click **Delete** to confirm deletion.

You can also click **Delete experiment** in the kebab menu of the relevant experiment row to delete an experiment.

The screenshot displays the H2O LLM Studio interface. On the left, the navigation pane has 'View experiments' selected. The main area shows a table of experiments with columns: name, dataset, metric, val metric, progress, status, and info. The table contains four rows: 'Experiment-2' (stopped, 3% progress), 'Experiment-1' (finished, 100% progress), 'Test-experiment' (finished, 100% progress), and 'H2O-LLM' (finished, 100% progress). A kebab menu is open for the 'H2O-LLM' experiment, showing options: 'New experiment', 'Rename experiment', 'Stop experiment', and 'Delete experiment' (highlighted). At the bottom, there are buttons for 'Refresh', 'Compare experiments', 'Stop experiments', and 'Delete experiments'.

name	dataset	metric	val metric	progress	status	info
Experiment-2	oasst	BLEU	0.0	3%	stopped	
Experiment-1	oasst	BLEU	3.8892	100%	finished	Runtime: 00:27:24
Test-experiment	oasst	BLEU	4.8871	100%	finished	Runtime
H2O-LLM	oasst	BLEU	4.5922	100%	finished	Runtime

4 of 4

Refresh Compare experiments Stop experiments Delete experiments

## Compare experiments

Using H2O LLM Studio, you can compare experiments and analyze how different model parameters affect model performance.

Follow the relevant steps below to compare experiments in H2O LLM Studio.

1. On the H2O LLM Studio left-navigation pane, click **View experiments**.
2. Click **Compare experiments**.
3. Select the experiments you want to compare.
4. Click **Compare experiments**.



The **Charts** tab visually represents the comparison of train/validation loss, metrics, and learning rate of selected experiments. The **Config** tab compares the configuration settings of selected experiments.

- **Note:** In addition, H2O LLM Studio also integrates with [Neptune](#), a powerful experiment tracking platform. By enabling Neptune logging when starting an experiment, you can easily track and visualize all aspects of your experiment in real time. This includes model performance, hyperparameter tuning, and other relevant metrics.

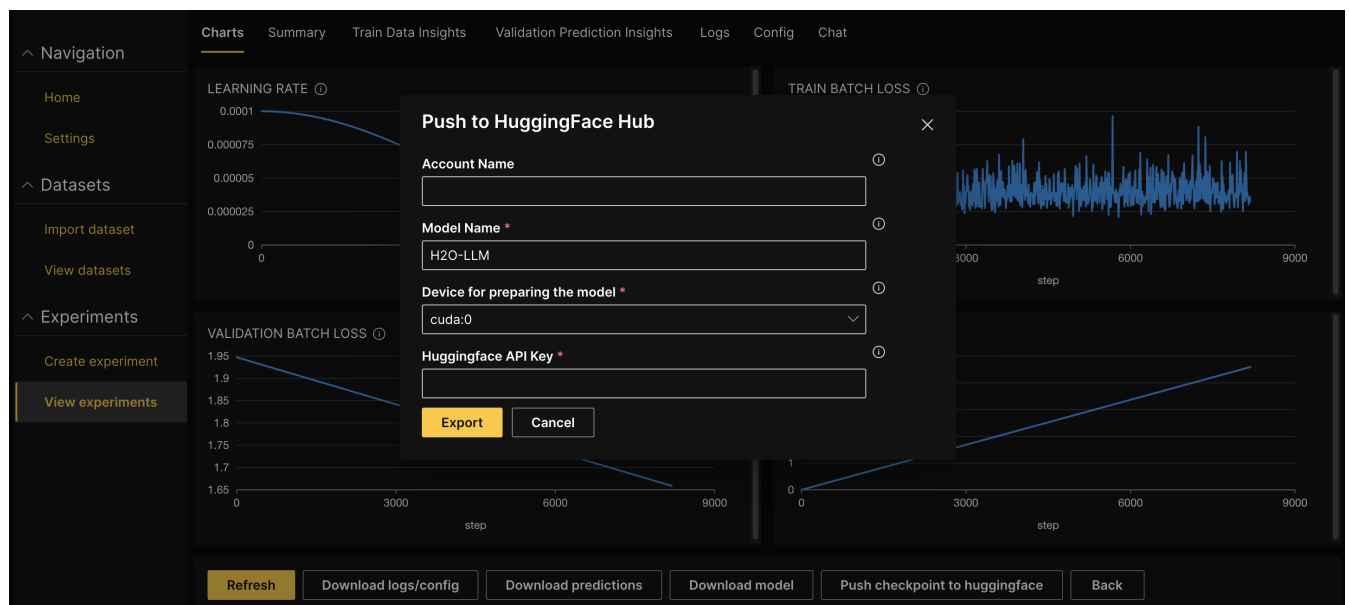
## Publish model to HuggingFace

If you are ready to share your trained model with a broader community, H2O LLM Studio allows you to export the fine-tuned model to [Hugging Face](#) with a single click.

- **Note:** Before publishing your model to the Hugging Face Hub, you need to have an API key with write access. To obtain an API token with write access, follow the [instructions provided by Hugging Face](#), which involve creating an account, logging in, and generating an access token with the appropriate permission.

To publish a trained model to Hugging Face Hub:

1. On the H2O LLM Studio left-navigation pane, click **View experiments**. You will see the experiments table with a list of all the experiments you have launched so far.
2. Click the name of the experiment that you want to export as a model.
3. Click **Push checkpoint to huggingface**.
4. Enter the **Account name** on Hugging Face to push the model to a particular account. Leaving it empty will push it to the default user account.
5. Enter the **Huggingface API Key** with write access.
6. Click **Export**.



## Download a model

Click **Download model** on the **View experiments** page to download the model locally.

Use the following code snippet to utilize the converted model in Jupyter Notebook or Google Colab.

```
from transformers import AutoModelForCausalLM, AutoTokenizer

model_name = "path_to_downloaded_model" # either local folder or Hugging Face model name

# Important: The prompt needs to be in the same format the model was trained with.
# You can find an example prompt in the experiment logs.
prompt = "<|prompt|>How are you?<|endoftext|><|answer|>"

tokenizer = AutoTokenizer.from_pretrained(model_name)
model = AutoModelForCausalLM.from_pretrained(model_name)
model.cuda().eval()

inputs = tokenizer(prompt, return_tensors="pt", add_special_tokens=False).to("cuda")
```

```

# generate configuration can be modified to your needs
tokens = model.generate(
    **inputs, # Input any question for the model. Ex: "What is the capital of USA?"
    max_new_tokens=256,
    temperature=0.3,
    repetition_penalty=1.2,
    num_beams=1
)[0]
tokens = tokens[inputs["input_ids"].shape[1]:]
answer = tokenizer.decode(tokens, skip_special_tokens=True)
print(answer)

```

You can enter any question for the model and change the parameters to get different outputs.

## Evaluate model using an AI judge

H2O LLM Studio provides the option to use an AI Judge like ChatGPT or a local LLM deployment to evaluate a fine-tuned model.

Follow the instructions below to specify a local LLM to evaluate the responses of the fine-tuned model.

1. Have an endpoint running of the local LLM deployment, which supports the OpenAI API format; specifically the [Chat Completions API](#).
2. Start the H2O LLM Studio server with the following environment variable that points to the endpoint.  
`OPENAI_API_BASE="http://111.111.111.111:8000/v1"`
3. Once H2O LLM Studio is up and running, click **Settings** on the left navigation panel to validate that the endpoint is being used correctly. The **Use OpenAI API on Azure** setting must be set to Off, and the environment variable that was set above should be the **OpenAI API Endpoint** value as shown below.

H2O LLM Studio v1.5.0

Navigation: Home, Settings, Datasets, Experiments

Settings

Default Experiment Settings

Azure Datalake container name

Kaggle username

Kaggle secret key

Number of Workers: 8

Logger: None

Neptune Project

Neptune API Token

Huggingface API Token

OpenAI API Token

GPT evaluation max samples: 100

Use OpenAI API on Azure: Off

OpenAI API Endpoint: http://111.111.111.111:8000/v1

OpenAI API Deployment ID: deployment-name

OpenAI API version: 2023-05-15

Experiment Maximum Settings


Number of Epochs: 50

Batch Size: 256

Save settings persistently Load settings Restore default settings

- **Note:** that changing the value of this field here on the GUI has no effect. This is only for testing the correct setting of the environment variable.
4. Run an experiment using **GPT** as the **Metric** and the relevant model name available at your endpoint as the **Metric Gpt Model**.





H2O LLM Studio  
v1.5.0

^ Navigation

Home
Settings

^ Datasets

Import dataset
View datasets

^ Experiments

Create experiment
View experiments

Skip Parent Probability
 0 ⓘ

Random Parent Probability
 0 ⓘ

Neptune Noise Alpha
 0 ⓘ

Prediction settings

Metric

GPT

 ⓘ

Metric Gpt Model

h2oai/h2o-danube-1.8b-chat

 ⓘ

Metric Gpt Template

general

 ⓘ

Min Length Inference
 2 ⓘ

Max Length Inference
 256 ⓘ

Max Time
 0 ⓘ

Batch Size Inference
 0 ⓘ

Do Sample ☒ Off ⓘ

Run experiment

- Validate that it is working as intended by checking the logs. Calls to the LLM judge should now be directed to your own LLM endpoint.

```

2024-05-07 15:31:37,788 - INFO: HTTP Request: POST http://:8080 /v1/chat/completions "HTTP/1.1 200 OK"
2024-05-07 15:31:37,906 - INFO: HTTP Request: POST http://:8080 /v1/chat/completions "HTTP/1.1 200 OK"

```

## Import a model to h2oGPT

Once the model has been fine-tuned using H2O LLM Studio, you can then use [h2oGPT](#) to query, summarize, and chat with your model.

The most common method to get the model from H2O LLM Studio over to h2oGPT, is to import it into h2oGPT via HuggingFace. However, if your data is sensitive, you can also choose to download the model locally to your machine, and then import it directly into h2oGPT.

You can use any of the following methods:

- Publish the model to HuggingFace and import the model from HuggingFace
- Download the model and import it to h2oGPT by specifying the local folder path
- Download the model and upload it to h2oGPT using the file upload option on the UI
- Pull a model from a Github repository or a resolved web link

### Steps

1. [Publish the model to HuggingFace](#) or [download the model locally](#).
2. If you opt to download the model, make sure you extract the downloaded .zip file.
3. Use the following command to import it into h2oGPT. `python generate.py --base_model=[link_or_path_to_folder]`

### Examples:

#### From HuggingFace

```
python generate.py --base_model=HuggingFaceH4/zephyr-7b-beta
```

#### From a Local File

```
python generate.py --base_model=zephyr-7b-beta.Q5_K_M.gguf
```

#### From a Repository

```
python generate.py --base_model=TheBloke/zephyr-7B-beta-AWQ
```

- **Note:** For more information, see the [h2oGPT documentation](#).

## FAQs

The sections below provide answers to frequently asked questions. If you have additional questions, please send them to [cloud-feedback@h2o.ai](mailto:cloud-feedback@h2o.ai).

---

### What are the general recommendations for using H2O LLM Studio?

The recommendation is to always start with the default settings. From there, the parameters that tend to have the largest impact are: - the LLM backbone - the number of epochs - the learning rate - the LoRA settings

- **Note:** For more information on experiment settings, see [Experiment Settings](#).

The parameters that have the largest impact on the amount of GPU memory being used are the [backbone dtype](#) and the [max length](#) (the length of the input sequence being used during model training).

- **Note:** For more information, see [this FAQ about GPU out-of-memory issues](#).

While these parameters will change the behavior of the fine-tuned model, the change that will be most impactful is the actual data used for fine tuning. Having clean data and enough samples (i.e., atleast 1000 records) is imperative.

---

### Is the tool multi-user or single user?

While it is possible for multiple users to use the same instance, the tool was created for a single user at a time.

---

### How can human feedback be applied in LLM Studio?

In order to apply human feedback to H2O LLM Studio, there is a problem type called DPO (Direct Preference Optimization), which is specifically used for learning human feedback. For these types of use cases, there would be a selected answer and a rejected answer column to train a reward model. This is a more stable form of the traditional RLHF. For more information, see [this paper about DPO](#) by Stanford University.

---

### How does H2O LLM Studio evaluate the fine-tuned model?

The valuation options are [BLEU](#), [Perplexity](#), and an AI Judge. For more information about the traditional NLP similarity metrics, see [BLEU](#) and [Perplexity](#) explained on the concepts page. You can also opt to use an AI judge by having an LLM model (ChatGPT or a local LLM) judge the performance of the response. This [sample prompt](#) is an example of a prompt that is used to have the LLM evaluate the response.

---

### Can I use a different AI Judge than ChatGPT?

Yes. For instructions on how to use a local LLM to evaluate the fine-tuned model, see [Evaluate model using an AI judge](#).

---

### How much data is generally required to fine-tune a model?

There is no clear answer. As a rule of thumb, 1000 to 50000 samples of conversational data should be enough. Quality and diversity is very important. Make sure to try training on a subsample of data using the “sample” parameter to see how big the impact of the dataset size is. Recent studies suggest that less data is needed for larger foundation models.

---

**Are there any recommendations for which backbone to use? Are some backbones better for certain types of tasks?**

The majority of the LLM backbones are trained on a very similar corpus of data. The main difference is the size of the model and the number of parameters. Usually, the larger the model, the better they are. The larger models also take longer to train. It is recommended to start with the smallest model and then increase the size if the performance is not satisfactory. If you are looking to train for tasks that are not directly question answering in English, it is also a good idea to look for specialized LLM backbones.

---

**What if my data is not in question-and-answer form and I just have documents? How can I fine-tune the LLM model?**

To train a chatbot style model, you need to convert your data into a question and answer format.

If you really want to continue pretraining on your own data without teaching a question-answering style, prepare a dataset with all your data in a single column Dataframe. Make sure that the length of the text in each row is not too long. In the experiment setup, remove all additional tokens (e.g. <|prompt|>, <|answer|>, for Text Prompt Start and Text Answer Start respectively) and disable **Add Eos Token To Prompt** and **Add Eos Token To Answer**. Deselect everything in the Prompt Column.

There are also other enterprise solutions from H2O.ai that may help you convert your data into a Q&A format. For more information, see [H2O.ai's Generative AI page](#) and this blogpost about [H2O LLM DataStudio: Streamlining Data Curation and Data Preparation for LLMs related tasks](#).

---

**Can the adapter be downloaded after fine-tuning so that the adapter can be combined with the backbone LLM for deployment?**

H2O LLM Studio provides the option to download only the LoRA adapter when a model was trained with LoRA. Once the experiment has finished running, click the **Download adapter** button to download the lora adapter\_weights separately from a fine-tuned model.

---

**I encounter GPU out-of-memory issues. What can I change to be able to train large models?**

There are various parameters that can be tuned while keeping a specific LLM backbone fixed. It is advised to choose 4bit/8bit precision as a backbone dtype to be able to train models  $\geq 7B$  on a consumer type GPU. [LORA](#) should be enabled. Besides that there are the usual parameters such as batch size and maximum sequence length that can be decreased to save GPU memory (please ensure that your prompt+answer text is not truncated too much by checking the train data insights).

---

**When does the model stop the fine-tuning process?**

The number of epochs are set by the user.

---

**What is the maximum dataset size that an LLM Studio instance can handle?**

The total dataset size is basically unlimited / only bound by disk space as all training is done in batches. There is no specific rule of thumb for maximum batch size - this depends strongly on backbone, context size, use of flash attention 2.0, use of gradient checkpointing, etc. We suggest using a batch size that just fills the RAM for maximum efficiency. While testing for maximum memory consumption, set padding quantile to 0. Make sure to set it back to 1 when you have found a good setting for the batch size to save on runtime.

---

## Where does H2O LLM Studio store its data?

By default, H2O LLM Studio stores its data in two folders located in the root directory in the app. The folders are named **data** and **output**. Here is the breakdown of the data storage structure: - **data/dbs**: This folder contains the user database used within the app. - **data/user**: This folder is where uploaded datasets from the user are stored. - **output/user**: All experiments conducted in H2O LLM Studio are stored in this folder. For each experiment, a separate folder is created within the **output/user** directory, which contains all the relevant data associated with that particular experiment. - **output/download**: Utility folder that is used to store data the user downloads within the app.

It is possible to change the default working directory of H2O LLM Studio by setting the `H2O_LLM_STUDIO_WORKDIR` environment variable. By default, the working directory is set to the root directory of the app.

---

## How can I update H2O LLM Studio?

To update H2O LLM Studio, you have two options:

1. Using the latest main branch: Execute the commands `git checkout main` and `git pull` to obtain the latest updates from the main branch.
2. Using the latest release tag: Execute the commands `git pull` and `git checkout v0.0.3` (replace 'v0.0.3' with the desired version number) to switch to the latest release branch.

The update process does not remove or erase any existing data folders or experiment records. This means that all your old data, including the user database, uploaded datasets, and experiment results, will still be available to you within the updated version of H2O LLM Studio.

Before updating, it is recommended to run the `git rev-parse --short HEAD` command and save the commit hash. This will allow you to revert to your existing version if needed.

---

## Once I have the [LoRA](#), what is the recommended way of utilizing it with the base model?

You can also export the LoRA weights. You may add them to the files to be exported [here](#). Before exporting, the LoRA weights are merged back into the original LLM backbone weights to make downstream tasks easier. You do not need to have PEFT, or anything else for your deployment.

---

## How to use H2O LLM Studio in Windows?

Use WSL 2 on Windows

---

## How can I easily fine-tune a large language model (LLM) using the command-line interface (CLI) of H2O LLM Studio when I have limited GPU memory?

If you have limited GPU memory but still want to fine-tune a large language model using H2O LLM Studio's CLI, there are alternative methods you can use to get started quickly.

- [Using Kaggle kernels](#)
  - [Using Google Colab](#)
- 

## Can I run a validation metric on a model post-training, optionally on a different validation dataset?

Yes.

1. After you have finished creating an experiment, click on the `more_vert` Kebab menu of the relevant experiment and select **New Experiment**.
2. Enable the **Use previous experiments weight** setting found at the top of the screen. This will now load the previous weights, and you can now change eval dataset, metric, and anything else as you see fit. To only do evaluation without any retraining, set the **Epochs** to 0.

---

### **What are the hardware/infrastructure sizing recommendations for H2O LLM Studio?**

When it comes to hardware requirements, it is important to note that the primary demand centers around the GPU and its associated VRAM. In terms of CPUs, most modern choices should suffice as NLP tasks typically do not heavily stress CPU performance. As for RAM, it's advisable to have a minimum of 128GB, with a stronger recommendation of 256GB or more, particularly when dealing with substantial model weights that must be accommodated in the CPU RAM.

---

## Key terms

H2O LLM Studio uses several key terms across its documentation, and each, in turn, is explained in the sections below.

### Prompt Engineering

Prompt engineering involves crafting precise and effective input queries to guide language models in generating desired outputs or responses.

### Agents

Software entities or components that interact with data or perform tasks within a system.

### ELO

An algorithm or method used to assess and rank the performance or accuracy of language models based on their proficiency in understanding and processing textual data.

### Vector Store

A Vector Store stores numerical representations of text for fast access in language models.

### Pre-training

The initial phase of training a machine learning model on a large dataset to learn general features before fine-tuning on a specific task.

### Attention

A mechanism that enables models to focus on specific parts of input data relevant to the task at hand, enhancing their understanding and performance.

### Embedding

Embedding refers to a mathematical representation of words or tokens in a numerical vector space, enabling machine learning models to understand and process language based on their context and relationships.

### Language Model

A language model is an AI system that understands and generates human language, predicting and generating text based on patterns and context within a given sequence of words.

### Transformer

A Transformer refers to a neural network architecture specifically designed for processing sequential data like text, using attention mechanisms to learn contextual relationships between words or tokens.

### Encoders and Decoders

Encoders and decoders are vital parts of sequence-to-sequence models used in natural language processing. Encoders process input data into a fixed-size representation, while decoders generate an output sequence based on that representation.

### Text generation

Text generation is the process of creating written content, such as sentences or paragraphs, using machine learning or AI algorithms based on patterns learned from existing text data.

### In-context learning

In-context learning refers to the process where a machine learning model continuously improves and adapts by considering the context of new information within its existing knowledge, enhancing its accuracy and understanding over time.

## **Few-shot learning**

Few-shot learning refers to a machine learning technique where a model can learn from a very small amount of labeled data to generalize and make predictions accurately on new, unseen data.

## **Summarization**

Summarization is the process of condensing a larger piece of text into a shorter, coherent version while retaining its essential information.

## **Fine-tuning**

Fine-tuning refers to adjusting and optimizing a pre-trained machine learning model using specific data to enhance its performance for a particular task.

## **GPT**

GPT stands for “Generative Pre-trained Transformer,” a type of language model that uses transformers to understand and generate human-like text based on vast amounts of training data.

## **GPU deployment**

GPU deployment is the utilization of graphics processing units (GPUs) to execute and accelerate the computations involved in deploying machine learning models, improving speed and efficiency in model inference or training.

## **Tokenization**

Tokenization is the process of breaking text into smaller units, typically words or phrases, to analyze or process them individually within a natural language processing system.