h2oGPT: Democratizing Large Language Models

Arno Candel, Jon McKinney, Philipp Singer, Pascal Pfeiffer, Maximilian Jeblick,
Prithvi Prabhu, Jeff Gambera, Mark Landry, Shivam Bansal, Ryan Chesler, Chun Ming Lee,
Marcos V. Conde, Pasha Stetsenko, Olivier Grellier, SriSatish Ambati [*]

H2O.ai, Inc.
Mountain View, CA

https://github.com/h2oai/h2ogpt
https://gpt.h2o.ai

https://github.com/h2oai/h2o-llmstudio

Abstract

Applications built on top of Large Language Models (LLMs) such as GPT-4 represent a revolution in AI due to their human-level capabilities in natural language processing. However, they also pose many significant risks such as the presence of biased, private, or harmful text, and the unauthorized inclusion of copyrighted material.

We introduce h2oGPT, a suite of open-source code repositories for the creation and use of LLMs based on Generative Pretrained Transformers (GPTs). The goal of this project is to create the world's best truly open-source alternative to closed-source approaches. In collaboration with and as part of the incredible and unstoppable open-source community, we open-source several fine-tuned h2oGPT models from 7 to 40 Billion parameters, ready for commercial use under fully permissive Apache 2.0 licenses. Included in our release is 100% private document search using natural language.

Open-source language models help boost AI development and make it more accessible and trustworthy. They lower entry hurdles, allowing people and groups to tailor these models to their needs. This openness increases innovation, transparency, and fairness. An open-source strategy is needed to share AI benefits fairly, and H2O.ai will continue to democratize AI and LLMs.

Keywords: *Natural language processing (NLP), Open Source, Generative Pretrained Transformer (GPT), Large Language Model (LLM), Hugging Face, Vector database, Chatbot, Document Search, LangChain, Commercial, Apache 2.0*

---

[*] Please cite this work as "h2oGPT by H2O.ai". This is work in progress. Correspondence regarding this technical report can be sent to `{arno, jon.mckinney, sri}@h2o.ai`

Contents

1 Introduction **3**
- 1.1 Why Open-Source LLMs? 3

2 The Making of h2oGPT **3**
- 2.1 Foundation Models and Datasets 3
  - 2.1.1 Pre-Training vs Fine-Tuning 3
  - 2.1.2 Foundation Models 4
  - 2.1.3 Foundation Datasets 5
- 2.2 Fine-Tuning 5
  - 2.2.1 Fine-Tuning Data Preparation 5
  - 2.2.2 H2O LLM Data Studio 6
  - 2.2.3 Fine-Tuning Methods 7
  - 2.2.4 Fine-Tuning Hardware requirements 10

3 Results **10**
- 3.1 The H2O.ai LLM Ecosystem 10
  - 3.1.1 h2oGPT models on Hugging Face 11
  - 3.1.2 ChatBot 12
  - 3.1.3 Private Document Chat 12
  - 3.1.4 No-Code Fine-Tuning with H2O LLM Studio 13
- 3.2 Validation, Limitations, and Capabilities 15
  - 3.2.1 Evaluation Metrics 15
  - 3.2.2 Current Weaknesses 15
  - 3.2.3 Current Capabilities 15

4 Outlook **16**

5 Conclusion **16**

6 Disclaimer **22**

Transparency and Accessibility

This is an open-source project, the code and models are publicaly available, free of charge.

The official GitHub repository for h2oGPT is `https://github.com/h2oai/h2ogpt`, and for LLM Studio is `https://github.com/h2oai/h2o-llmstudio`, both are open to contributions from the community and in constant evolution.

The foundation large language models (LLMs) presented in this work, can be tested in our online playground `https://gpt.h2o.ai/` — no login required, completely free.

1 Introduction

Recent advances in LLMs and GPTs are all over the news. Companies like OpenAI, Google, Anthropic, Microsoft, Cohere, Meta, Stability.AI, AI21 Labs, and many others have established leadership in the development and integration of LLMs. However, none of the above companies are providing truly open-source commercially viable models or even training data.

H2O.ai has built several world-class Machine Learning, Deep Learning and AI platforms over the past decade, much of it as open-source software (and on top of existing open-source software), and has earned the trust of its customers across the globe. We are ideally positioned to provide an open-source GPT ecosystem to enterprises, organizations, and individuals across the world.

1.1 Why Open-Source LLMs?

Every nation, state, and city needs its own GPT. This is because LLMs can be used for a variety of purposes, such as health care, science, and education.

While commercially hosted and centralized LLMs like OpenAI's ChatGPT/GPT-4, Anthropic's Claude, Microsoft's Bing AI Chat, Google's Bard, and Cohere are powerful and effective, they have certain limitations compared to open-source LLMs:

- **Data Privacy and Security**: Using hosted LLMs requires sending data to external servers. This can raise concerns about data privacy, security, and compliance, especially for sensitive information or industries with strict regulations.
- **Dependency and Customization**: Hosted LLMs often limit the extent of customization and control, as users rely on the service provider's infrastructure and predefined models. Open-source LLMs allow users to tailor the models to their specific needs, deploy on their own infrastructure, and even modify the underlying code.
- **Cost and Scalability**: Hosted LLMs usually come with usage fees, which can increase significantly with large-scale applications. Open-source LLMs can be more cost-effective, as users can scale the models on their own infrastructure without incurring additional costs from the service provider.
- **Access and Availability**: Hosted LLMs may be subject to downtime or limited availability, affecting users' access to the models. Open-source LLMs can be deployed on-premises or on private clouds, ensuring uninterrupted access and reducing reliance on external providers.

Overall, open-source LLMs offer greater flexibility, control, and cost-effectiveness, while addressing data privacy and security concerns. They foster a competitive landscape in the AI industry and empower users to innovate and customize models to suit their specific needs.

2 The Making of h2oGPT

In this section, we detail some of the work done to create the fine-tuned h2oGPT models we released. We show what data and models were used in the process. More detail can be found on h2oGPT GitHub issues and H2O LLM Studio GitHub issues.

2.1 Foundation Models and Datasets

To create a conversational GPT, we need a foundation model that can generate tokens, and we need to fine-tune it to become conversational (i.e., create useful answers for given prompts). One can also fine-tune a foundation model to become good at summarizing articles, or good at converting articles into JSON key/value pairs etc., but the key is a good foundation model and a small but high-quality dataset for fine-tuning.

2.1.1 Pre-Training vs Fine-Tuning

- **Pre-training**: Typically on TBs of data, gives the LLM the ability to master one or many **languages**. Pre-training usually takes weeks or months on dozens or hundreds of GPUs. The most common concern is underfitting and cost.

		Humanities	STEM	Social Sciences	Other	Average
GPT-NeoX (**h2oGPT**)	20B	29.8	34.9	33.7	37.7	33.6
Falcon (**h2oGPT**)	40B					54.2
GPT-3	175B	40.8	36.7	50.4	48.8	43.9
GPT-4	?					**86.4**
Gopher	280B	56.2	47.4	71.9	66.1	60.0
Chinchilla	70B	63.6	54.9	79.3	**73.9**	67.5
PaLM	8B	25.6	23.8	24.1	27.8	25.4
	62B	59.5	41.9	62.7	55.8	53.7
	540B	**77.0**	**55.6**	**81.0**	69.6	69.3
LLaMa	7B	34.0	30.5	38.3	38.1	35.1
	13B	45.0	35.8	53.8	53.3	46.9
	33B	55.8	46.0	66.7	63.4	57.8
	65B	61.8	51.7	72.9	67.4	63.4

Table 1: **Massive Multitask Language Understanding (MMLU).** Five-shot accuracy. From LLaMa paper. Falcon value from h2oGPT repository. GPT-4 value from GPT-4 TR.

- **Fine-tuning**: Typically on MBs or GBs of data, makes a model more familiar with a **specific style** of prompting, which generally leads to improved outcomes for this one specific case. The most common concern is overfitting. Fine-tuning usually takes hours or days on a few GPUs.

2.1.2 Foundation Models

The following permissively licensed foundation models are available currently (May 2023), in Hugging Face format, for easy adoption:

- EleutherAI/pythia-6.9b
- EleutherAI/pythia-12b and EleutherAI/pythia-12b-deduped
- **EleutherAI/gpt-neox-20b**
- mosaicml/mpt-7b-storywriter
- tiiuae/falcon-7b
- **ttiuae/falcon-40b**
- bigscience/bloom

The largest foundation models we used were GPT-NeoX-20B: An Open-Source Autoregressive Language Model (from April 2022), and Falcon-40B (from May 2023). The largest available fully open-source model to this day is Bloom 176B, but it is too big to be practical, and also undertrained. The above models from EleutherAI and bigscience were trained on a relatively small number of tokens using Chinchilla scaling laws, but it later turned out that smaller models trained on more tokens can perform even better, such as LLaMa, and now Falcon. The above models (except for mpt-7b-storywriter) also have relatively short context lengths of only 2048 tokens (can only summarize about one page), and models with larger context lengths would be preferable for many downstream tasks.

Table 1 shows the placement of h2oGPT in the ecosystem of non-open-source models.

Several efforts by the open-source community are underway to train improved fully open-source permissive (Apache 2.0 license or similar) foundation models:

- Open LLaMa
- Red Pajama
- MosaicML MPT-7B

We are not currently training our own foundation models, as more community-driven architectural improvements are likely to arrive soon to further improve the performance of the models. Every small architectural change will require training from scratch.

2.1.3 Foundation Datasets

All above models (except for Falcon models) were trained on the Pile dataset, 825 GiB of data. This dataset contains some questionable content, as it was sourced from the internet, but the data preparation methods and the dataset are publicly available. Falcon models were trained on the RefinedWeb dataset, which is 2.8 TiB of internet data prepared with enhanced filtering and deduplication methods.

Several efforts are underway to improve the training data for future foundation models:

- Pile V2
- Red Pajama

2.2 Fine-Tuning

Given a suitable foundation model (currently with 7, 12, 20 or 40 billion parameters), we need a fine-tuning dataset and a Linux box with suitable GPUs. More information about fine-tuning is on our GitHub pages.

2.2.1 Fine-Tuning Data Preparation

To fine-tune a model, we typically need to provide both input (i.e., a prompt) and output (i.e., the response). As most data is not in this format, some effort is required to create these pairs. One example of this:

```
{
  "input": "Who are you?",
  "output": "I am h2oGPT.",
}
```

Obviously, more sophisticated inputs and outputs will lead the model to become more sophisticated, to the degree that the foundation model's abilities allow. If all outputs in the fine-tuning dataset are very brief, then the model will learn to be brief in general. Note that the model will not be able to memorize all the fine-tuning input/output pairs, but mostly learn the **style** of the fine-tuning data.

In addition to creating input/output pairs, several other things have to be taken care of:

- Prompt engineering (e.,g., injection of `<human>:` and `<bot>:` into the text to indicate input/outputs)
- data pre-processing (e.g., removing incomplete sentences, too long dialogues)
- data filtering (e.g., removing profanity or undesired responses that are too short or low quality)

So far, we have created over 1800 lines of code for data processing and cleaning, required for efficient fine-tuning. All the code is available at our GitHub repository.

Most of our h2oGPT models have been trained on at least a significant part of the recently open-sourced Open Assistant dataset. It contains high-quality conversational data, sourced from crowd-sourced humans. While the quality isn't as high as e.g., GPT-4 output (which we can't use due to the terms of service), it's the best open-source instruction data available at the moment. The challenge lies in identifying low-quality data points, or in the generation of even higher-quality data points from it, see our roadmap items below.

We use the Open Assistant Reward Model to assign scores to each conversation, and we drop those with low scores.

For fine-tuning our models, we end up with high-quality conversations like these (shown here in fully formatted form for actual training, with `"<human>: "` and `"<bot>: "` prefixes to indicate inputs and outputs).

```
"<human>: What is the historical origin of werewolf stories? <bot>: The historic
starting place of werewolf tales may be traced again to historic civilizations,
consisting of the Greeks and Romans, who informed stories of people who converted
into wolves. Werewolf legend additionally has roots in European folklore, in which
it became believed that people may want to change into wolves via a curse or via
way of means of being bitten via way of means of any other werewolf. The notion
of werewolves continued via the Middle Ages and have become a famous problem in
Gothic literature at some point of the Renaissance. <human>: Why is it said that
silver is needed to kill a werewolf? <bot>: Now, silver sulfide is not shown to
be toxic to any other animals, but it's not soluble in water, and so can only be
ingested. In a werewolf, it would lift from the silver and travel through the
bloodstream of the animal, blocking off blood vessels and poisoning cells. And
it is this that kills the werewolf. <human>:"
```

The `"<human>:"` at the end indicates that the response started by the last `"<bot>:"` is complete. This way, the model learns to properly terminate its responses. Note that the choice of chat-indicators like `"<human>:"` is arbitrary, and dedicated tokens are often used to avoid tokenization ambiguities.

Training on realistic multi-turn conversations like the one shown above substantially improves generation quality for a chatbot, according to common sense and other research such as LIMA. During a chat conversation, prior dialogues are added as part of the context at generation time, so follow-up prompts like `Summarize the above` are perfectly fine.

We make this dataset available on our Hugging Face page. It is intended to be used in combination with the fine-tuning methods provided by the h2oGPT repository.

2.2.2 H2O LLM Data Studio

We also improved the foundational scripts used in the data preparation for the h2oGPT model. We added more generalization in the code, comprehensive error handling, handling a variety of training/tuning tasks, and a variety of text cleaning and data preparation utility functions. This led to the development of H2O LLM Data Studio - a toolkit for data preparation for LLM fine-tuning.

LLM Data Studio can be used to prepare datasets for a variety of downstream tasks, This includes:

- **Question Answering:** It involves preparing datasets that consist of contextual information, questions, and corresponding answers. This task is essential for training question-answering models that can accurately respond to queries based on the provided context. The dataset preparation process focuses on building a well-structured dataset for training question-answering systems.
- **Text Summarization:** It involves preparing datasets that consist of articles and their corresponding summaries. In this task, the dataset preparation process focuses on extracting important information from the articles and creating concise summaries that capture the key points. With the prepared datasets, users can train text summarization models to generate concise and informative summaries from longer pieces of text.
- **Instruct Tuning:** It involves preparing datasets that consist of prompts or instructions and their corresponding responses. This task is essential for training models that effectively understand and adhere to the provided instructions and accurately respond to user prompts.
- **Human-Bot Conversations:** It involves preparing datasets that contain multiple conversations between human users and chat bots. This task is essential for training models that can understand user intents, and provide accurate responses, leading to improved conversational experiences. During dataset preparation, the focus is on structuring and organizing the conversational data, including user queries, bot responses, and any relevant context.
- **Continued Pre-Training:** It involves preparing datasets with long texts to facilitate further pre-training of language models. In this task, the dataset preparation process focuses on organizing long textual data to allow the language models to learn from extensive and diverse linguistic patterns, leading to enhanced language understanding and generation capabilities.

Key techniques supported in LLM Data Studio:

- Data Augmentation: Augment or mix multiple data sets as a single data object
- Text Cleaning: Clean the text using different cleaning methods such as stop words removal, punctuation removal, special character removal, case handling
- Profanity Check: Check and remove any texts objects having profanity
- Text Quality Check: Check and remove any texts having profanity
- Truncate by Length: Truncate the sentence based on a max length parameter
- Valid Q&A: Calculate the similarity score and filter the dataset based on a similarity threshold
- Pad Sequence: Pad the sequence based on a maximum length parameter
- Truncate Sequence by Score: Truncate the sequence based on a score and max length parameter required for the model.
- Output Conversion: Convert the transformed dataset to an output object such as JSON
- Compression Ratio Filter: Filter the text summarizing by comparing the compression ratio of the summaries
- Boundary Marking: Add start and end tokens in the boundaries of the summary text

The typical workflow for data preparation in H2O LLM Studio involves several sequential steps. Firstly, the user performs data ingestion, where they import various types of documents from different connectors. Once the data is ingested, the next step is to select the target training task, which can include tasks like continued pretraining, instruct tuning, chatbot development, or RLHF protection.

After selecting the training task, users have the option to augment their dataset by incorporating additional data from other sources. This data mix-in or augmentation step allows for the enrichment of the existing dataset.

Subsequently, the data cleaning process takes place, wherein low-quality parts of the data are removed. This includes eliminating problematic elements like long lines of pure spaces or unusual characters that may hinder analysis or modeling.

To ensure data quality, a data quality checking step is implemented. This involves employing techniques like bleu/meteor/similarity or RLHF reward models to identify and filter out data with poor quality. Additional filters, such as length-based filtering (e.g., short concise answers vs. long answers), and checks for profanity can also be applied during this stage.

Once the text has been cleaned and verified for quality, the user selects the target tool for data transformation. This step involves converting the data, along with its associated metadata, into a suitable format such as JSON for utilization in LLM Studio, h2oGPT, or any other target tool.

Lastly, the data is prepared for the target model. Different models may have specific requirements for context length or cutoff length, and the data needs to be adjusted accordingly. This ensures that the text is appropriately truncated to match the desired specifications of the target model, avoiding any truncation issues or poor data representation.

By following this systematic workflow, users can effectively prepare their data for analysis and modeling in H2O LLM Studio, facilitating accurate and reliable research outcomes.

H2O LLM Data Studio is also part of the H2O LLM Ecosystem and is made available to users for the purpose of data cleaning and preparation for fine-tuning LLMs.

2.2.3 Fine-Tuning Methods

LoRA We use Huggingface PEFT and its implementation of LoRA (Low Rank Approximation) LoRA. This results in substantial speed-up and lower memory use compared to full fine-tuning. Only as a small fraction of weights are trainable, and the required optimizer state is of the order of 20MB instead of 20GB, reducing the memory footprint by at least a factor of 2, and leading to measurable speedups as fewer GPUs are needed and fewer gradients need to be computed. In addition, full fine-tuning can result in catastrophic forgetfulness, which can be prevented using adapter methods like LoRA by focusing the fine-tuning on specific parts of the neural network architecture, such as the attention heads.

Injecting LoRA into linear layers turns the dense matrices into read-only weights, and adds a product of two small trainable matrices with a scaling factor, for reduced memory overhead during back-propagation (training).

Original model architecture for the `h2oai/h2ogpt-oasst1-falcon-40b` model:

```
RWForCausalLM(
  (transformer): RWModel(
    (word_embeddings): Embedding(65024, 8192)
    (h): ModuleList(
      (0-59): 60 x DecoderLayer(
        (ln_attn): LayerNorm((8192,), eps=1e-05, elementwise_affine=True)
        (ln_mlp): LayerNorm((8192,), eps=1e-05, elementwise_affine=True)
        (self_attention): Attention(
          (maybe_rotary): RotaryEmbedding()
          (query_key_value): Linear(in_features=8192, out_features=9216, bias=False)
          (dense): Linear(in_features=8192, out_features=8192, bias=False)
          (attention_dropout): Dropout(p=0.0, inplace=False)
        )
        (mlp): MLP(
          (dense_h_to_4h): Linear(in_features=8192, out_features=32768, bias=False)
          (act): GELU(approximate='none')
          (dense_4h_to_h): Linear(in_features=32768, out_features=8192, bias=False)
        )
      )
    )
    (ln_f): LayerNorm((8192,), eps=1e-05, elementwise_affine=True)
  )
  (lm_head): Linear(in_features=8192, out_features=65024, bias=False)
)
```

After adding LoRA adapters for the `Linear` layers (dense matrix multiplies), we get the following model architecture for the trainable weights:

```
PeftModelForCausalLM(
  (base_model): LoraModel(
    (model): RWForCausalLM(
      (transformer): RWModel(
        (word_embeddings): Embedding(65024, 8192)
        (h): ModuleList(
          (0-59): 60 x DecoderLayer(
            (ln_attn): LayerNorm((8192,), eps=1e-05, elementwise_affine=True)
            (ln_mlp): LayerNorm((8192,), eps=1e-05, elementwise_affine=True)
            (self_attention): Attention(
              (maybe_rotary): RotaryEmbedding()
              (query_key_value): Linear8bitLt(
                in_features=8192, out_features=9216, bias=False
                (lora_dropout): ModuleDict(
                  (default): Dropout(p=0.05, inplace=False)
                )
                (lora_A): ModuleDict(
                  (default): Linear(in_features=8192, out_features=8, bias=False)
                )
                (lora_B): ModuleDict(
                  (default): Linear(in_features=8, out_features=9216, bias=False)
                )
                (lora_embedding_A): ParameterDict()
                (lora_embedding_B): ParameterDict()
              )
              (dense): Linear8bitLt(
                in_features=8192, out_features=8192, bias=False
                (lora_dropout): ModuleDict(
                  (default): Dropout(p=0.05, inplace=False)
                )
                (lora_A): ModuleDict(
                  (default): Linear(in_features=8192, out_features=8, bias=False)
                )
                (lora_B): ModuleDict(
                  (default): Linear(in_features=8, out_features=8192, bias=False)
                )
                (lora_embedding_A): ParameterDict()
                (lora_embedding_B): ParameterDict()
              )
              (attention_dropout): Dropout(p=0.0, inplace=False)
            )
            (mlp): MLP(
              (dense_h_to_4h): Linear8bitLt(
                in_features=8192, out_features=32768, bias=False
                (lora_dropout): ModuleDict(
                  (default): Dropout(p=0.05, inplace=False)
                )
                (lora_A): ModuleDict(
                  (default): Linear(in_features=8192, out_features=8, bias=False)
                )
                (lora_B): ModuleDict(
                  (default): Linear(in_features=8, out_features=32768, bias=False)
                )
                (lora_embedding_A): ParameterDict()
                (lora_embedding_B): ParameterDict()
              )
              (act): GELU(approximate=’none’)
              (dense_4h_to_h): Linear8bitLt(
                in_features=32768, out_features=8192, bias=False
                (lora_dropout): ModuleDict(
                  (default): Dropout(p=0.05, inplace=False)
                )
                (lora_A): ModuleDict(
                  (default): Linear(in_features=32768, out_features=8, bias=False)
                )
                (lora_B): ModuleDict(
                  (default): Linear(in_features=8, out_features=8192, bias=False)
                )
                (lora_embedding_A): ParameterDict()
                (lora_embedding_B): ParameterDict()
              )
            )
          )
        )
        (ln_f): LayerNorm((8192,), eps=1e-05, elementwise_affine=True)
      )
      (lm_head): Linear(in_features=8192, out_features=65024, bias=False)
    )
  )
)
trainable params: 55541760 || all params: 41358835712 || trainable%: 0.13429236835089367
```

The resulting number of trainable parameters is typically around 0.1% of the original weights, and the degree of approximation can be parameterized with several tuning parameters, most of which don’t seem to have a large impact on accuracy, which is great. This makes LoRA one of the most useful techniques for efficient fine-tuning.

bitsandbytes To further reduce memory requirements on costly GPU hardware, we make use of 16-bit, 8-bit or 4-bit training using mixed precision hardware and software support, instead of 32-bit or 64-bit precision, which are commonly used across most computing applications. The benefit of the speedup and cost savings from being able to fit the entire model into one GPU is much higher than the downside due to loss of precision. Training or inference with the base model in 8-bit or 4-bit is achieved using PEFT and bitsandbytes. While this lowers the memory cost by about a factor of two compared to the use of LoRA alone, it is substantially slower for training than 16-bit on current architectures. Training using 4-bit precision was just made possible and should help with further democratizing LLM fine-tuning to consumer GPUs with 24GB of VRAM or less, cf QLoRA.

Native training using 8-bit floating point precision developed by NVIDIA on H100 GPUs should lead to significant memory savings without compromising training speed, but we haven't had a chance to try that yet.

2.2.4 Fine-Tuning Hardware requirements

NVIDIA GPUs Using LoRA and 8-bit training, we can fine-tune LLMs with 20B parameters on commodity GPUs with 24GB of VRAM, but just barely, and only for short input/outputs (token length), with batch size 1. We recommend A100 or A6000 (Ada) NVIDIA cards for fine-tuning, or H100, to get the best price/performance, or the use of 4-bit training for cards with less VRAM.

These are the minimally recommended GPU memory sizes for fine-tuning the respective h2oGPT models and 16-bit training is recommended wherever possible, as it can be much faster (by a factor 4 over 8-bit, 4-bit performance is not yet widely tested):

h2oGPT Model Size	4-bit	8-bit	16-bit
7B	16GB	12GB	16GB
12B	16GB	24GB	32GB
20B	16GB	32GB	48GB
30B (research)	24GB	48GB	80GB
40B	48GB	80GB	2x80GB
65B (research)	48GB	80GB	2x80GB

Table 2: h2oGPT model size comparison.

16GB/32GB cards include V100, 24GB cards include 3090/4090, 40GB cards include A100, 48GB cards include A6000/A6000 Ada, 80GB cards include A100/H100.

Training on multiple GPUs is always faster than training on one GPU, and data parallelism is enabled by default. Larger GPU memory sizes can allow faster training too, since more training data can be streamed. For example, if the model requires 20GB of memory, then one 80GB GPU might allow a batch size of 8, while a 24GB card can only fit a batch size of 1. Having 8x80GB can hence lead to a significant speedup compared to 1x24GB etc. Multi-node multi-GPU training is also possible in the existing framework, and LoRA training requires minimal communication between nodes, which makes it feasible to train on nodes with low interconnect speeds.

We did not try fine-tuning with TPUs or other accelerators, as NVIDIA GPUs are currently the best-supported most available hardware.

3 Results

Using the methods outlined above, our makers at H2O.ai have created suitable fine-tuning datasets, prompt engineering techniques, fine-tuning methods, UIs, chatbots, and VectorDB-based private document chat systems, and we are open-sourcing everything.

3.1 The H2O.ai LLM Ecosystem

Our open-source LLM ecosystem currently includes the following components:

- **Code, data, and models**: Fully permissive, commercially usable code, curated fine-tuning data, and fine-tuned models ranging from 7 to 20 billion parameters.
- **State-of-the-art fine-tuning**: We provide code for highly efficient fine-tuning, including targeted data preparation, prompt engineering, and computational optimizations to fine-tune LLMs with up to 20 billion parameters (even larger models expected soon) in hours on commodity hardware or enterprise servers. Techniques like low-rank approximations (LoRA) and data compression allow computational savings of several orders of magnitude.

- **Chatbot**: We provide code to run a multi-tenant chatbot on GPU servers, with an easily shareable end-point and a Python client API, allowing you to evaluate and compare the performance of fine-tuned LLMs.
- **Document Chat using VectorDB**: We provide code for a fully functional natural language-based document search system using Vector databases and prompt engineering. Of course, 100% private, and no internet connection is needed.
- **H2O LLM Studio**: Our no-code LLM fine-tuning framework created by the world's top Kaggle grandmasters makes it even easier to fine-tune and evaluate LLMs. H2O LLM Studio democratizes LLMs for everyone. This means that anyone can use H2O LLM Studio to fine-tune large open-source LLMs like h2oGPT and others on their own private data and on their servers.

The links to our open-source repositories and discussion channels are:

- h2oGPT `https://github.com/h2oai/h2ogpt`
- H2O LLM Studio `https://github.com/h2oai/h2o-llmstudio`
- H2O.ai on Hugging Face `https://huggingface.co/h2oai`
- H2O.ai Generative Discord Channel

Everything we release is based on fully permissive data and models (exceptions such as LLaMa-based models are explicitly marked as research only), with all code open-sourced, enabling broader access for businesses and commercial products without legal concerns, thus expanding access to cutting-edge AI while adhering to licensing requirements.

3.1.1 h2oGPT models on Hugging Face

We are making our models available on the Hugging Face repository. Notable models include:

- `h2oai/h2ogpt-oasst1-falcon-40b`
- `h2oai/h2ogpt-oig-oasst1-falcon-40b`
- `h2oai/h2ogpt-oasst1-512-20b`
- `h2oai/h2ogpt-oasst1-512-12b`
- `h2oai/h2ogpt-oig-oasst1-512-6_9b`
- `h2oai/h2ogpt-gm-oasst1-en-2048-falcon-40b-v1`
- `h2oai/h2ogpt-gm-oasst1-en-1024-20b`
- `h2oai/h2ogpt-gm-oasst1-en-2048-falcon-7b-v2`
- `h2oai/h2ogpt-research-oasst1-512-30b` (non-commercial)
- `h2oai/h2ogpt-research-oasst1-512-65b` (non-commercial)

To use the models from Python is easy:

```
!pip install transformers==4.29.2
!pip install accelerate==0.19.0
!pip install torch==2.0.1
!pip install einops==0.6.1

import torch
from transformers import pipeline, AutoTokenizer
tokenizer = AutoTokenizer.from_pretrained("h2oai/h2ogpt-oasst1-falcon-40b",
padding_side="left")
generate_text = pipeline(model="h2oai/h2ogpt-oasst1-falcon-40b",
    tokenizer=tokenizer, torch_dtype=torch.bfloat16, trust_remote_code=True,
    device_map="auto", prompt_type="human_bot")

res = generate_text("Why is drinking water so healthy?", max_new_tokens=100)
```

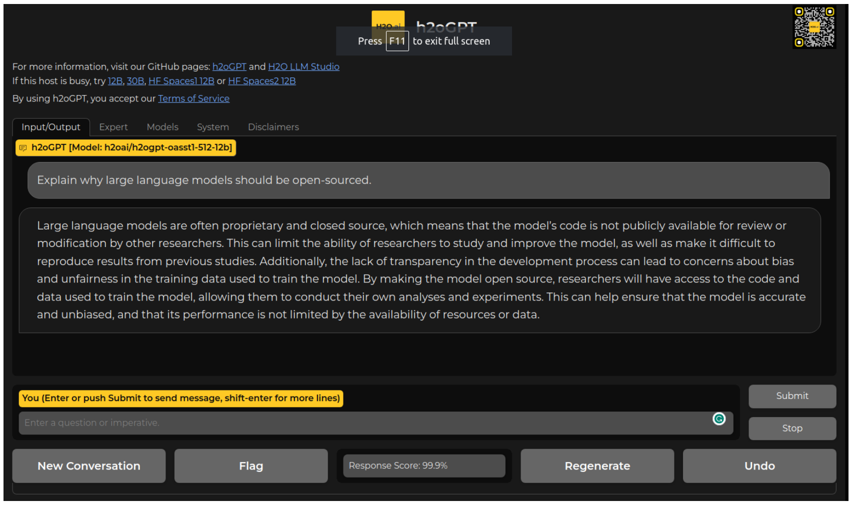
```
print(res[0]["generated_text"])

>>> Drinking water is healthy because it helps to keep your body hydrated and functioning
>>> properly. It also helps to flush out toxins and waste from the body, which can help
>>> to improve your overall health. Additionally, drinking water can help to regulate
>>> your body temperature, which can help to prevent dehydration and heat exhaustion.
```

3.1.2 ChatBot

h2oGPT `https://github.com/h2oai/h2ogpt` contains a simple chatbot GUI and client/server API based on Gradio.

```
python generate.py --base_model=h2oai/h2ogpt-oasst1-512-12b
```


h2oGPT
Press F11 to exit full screen
Input/Output
Expert
Models
System
Disclaimers
h2oGPT [Model: h2oai/h2ogpt-oasst1-512-12b]
You (Enter or push Submit to send message, shift-enter for more lines)
Enter a question or imperative.
Submit
Stop
New Conversation
Flag
Regenerate
Undo

For more information, visit our GitHub pages: h2oGPT and H2O LLM Studio
If this host is busy, try 12B, 30B, HF Spaces1 12B or HF Spaces2 12B
By using h2oGPT, you accept our Terms of Service

Explain why large language models should be open-sourced.

Large language models are often proprietary and closed source, which means that the model's code is not publicly available for review or modification by other researchers. This can limit the ability of researchers to study and improve the model, as well as make it difficult to reproduce results from previous studies. Additionally, the lack of transparency in the development process can lead to concerns about bias and unfairness in the training data used to train the model. By making the model open source, researchers will have access to the code and data used to train the model, allowing them to conduct their own analyses and experiments. This can help ensure that the model is accurate and unbiased, and that its performance is not limited by the availability of resources or data.

Response Score: 99.9%

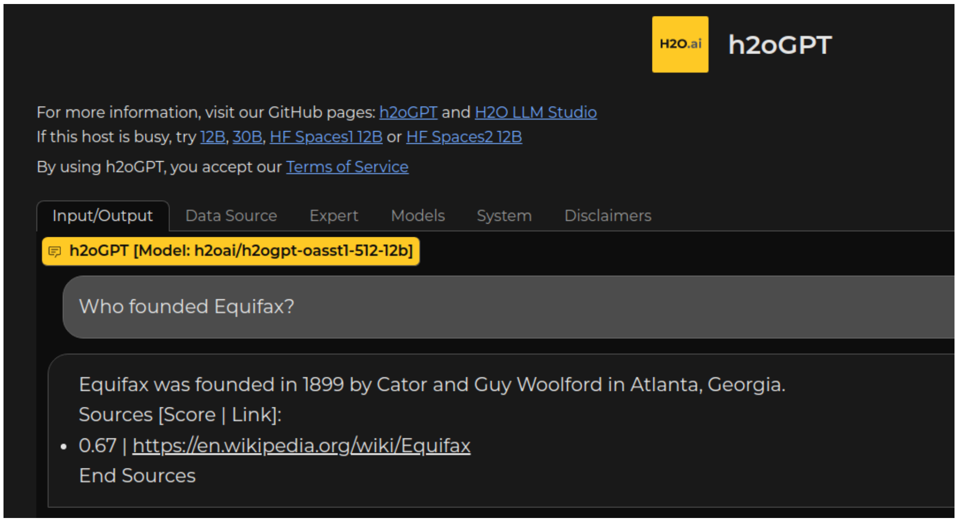
Chatbot features include:

- supports any open-source LLM from Hugging Face
- offline mode with no internet access required
- comparison of any 2 models
- supports LoRA adapter weights on top of any LLM
- multi-GPU sharding
- automatic scoring of responses using a reward model trained on human feedback
- 4-bit quantization options
- automatic expansion of context from multiple back-and-forth conversations

3.1.3 Private Document Chat

It is well-known that LLMs can hallucinate or confabulate their responses, c.f. On the Dangers of Stochastic Parrots. It is an active area of research to understand under what conditions this occurs and how to contain it. One way to ground LLMs is to provide source content as context for any query. The query and source content are embedded and similarity is estimated using a vector database. h2oGPT includes FAISS in-memory and Chroma persistent vector databases, relying upon instruct-tuned LLMs to answer the question given the context of top `k` chunks of source content.

```
python generate.py --base_model=h2oai/h2ogpt-research-oasst1-512-30b
  --langchain_mode=wiki_full
```



For more information, visit our GitHub pages: h2oGPT and H2O LLM Studio
If this host is busy, try 12B, 30B, HF Spaces1 12B or HF Spaces2 12B
By using h2oGPT, you accept our Terms of Service

h2oGPT [Model: h2oai/h2ogpt-oasst1-512-12b]

Who founded Equifax?

Equifax was founded in 1899 by Cator and Guy Woolford in Atlanta, Georgia.
Sources [Score | Link]:
- 0.67 | https://en.wikipedia.org/wiki/Equifax

End Sources

Document chat features include:

- fact-based question answering for documents
- 20GB Wikipedia state is pre-loaded
- offline mode with no internet access required
- persistent database with vector embeddings
- ability to ingest various document types

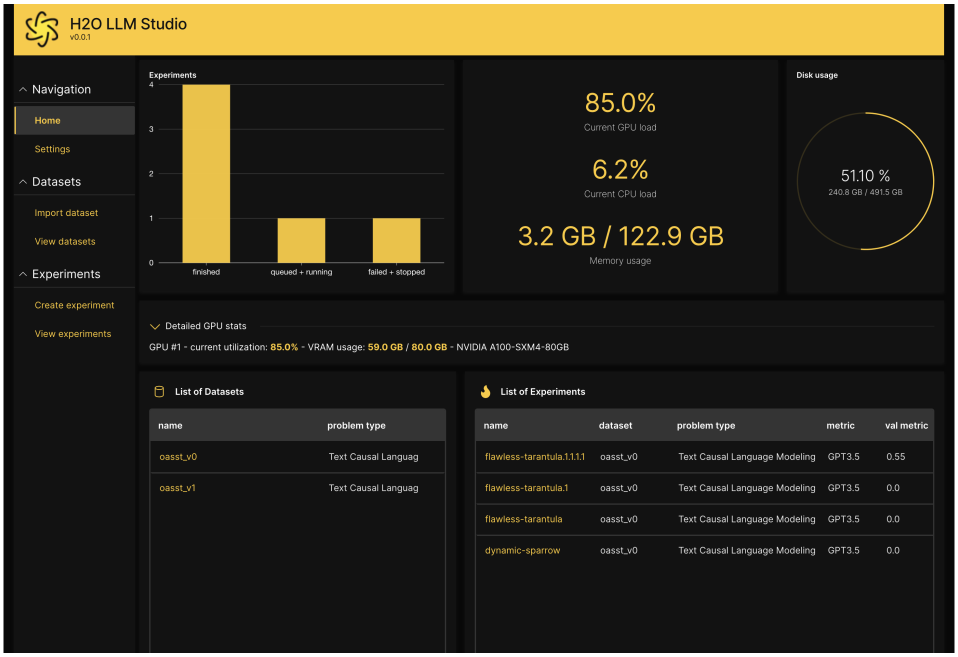
3.1.4 No-Code Fine-Tuning with H2O LLM Studio

H2O LLM Studio `https://github.com/h2oai/h2o-llmstudio` is an open-source framework that offers both a no-code graphical user interface (GUI) and a command-line interface (CLI) for fine-tuning LLMs. It allows users to train and tweak state-of-the-art LLMs with a variety of hyperparameters, without requiring any coding experience. It supports various advanced finetuning techniques such as Low-Rank Adaptation (LoRA) and 8-bit model training with a low memory footprint. The software allows users to track and compare model performance visually and provides an option to chat with the model for instant performance feedback. Additionally, it facilitates easy model export to the Hugging Face Hub for sharing with the community.

The latest updates to H2O LLM Studio include storing experiment configurations in YAML format and added functionality for supporting nested conversations in data. The system requirements include Ubuntu 16.04+ and an NVIDIA GPU with driver version >= 470.57.02. The software also supports Docker for easy deployment, and it expects CSV input with at least two columns - one for the instruct column and another for the model's expected answer.

Starting H2O LLM Studio is easy:

```
make wave
```



85.0%
Current GPU load

6.2%
Current CPU load

3.2 GB / 122.9 GB
Memory usage

Detailed GPU stats

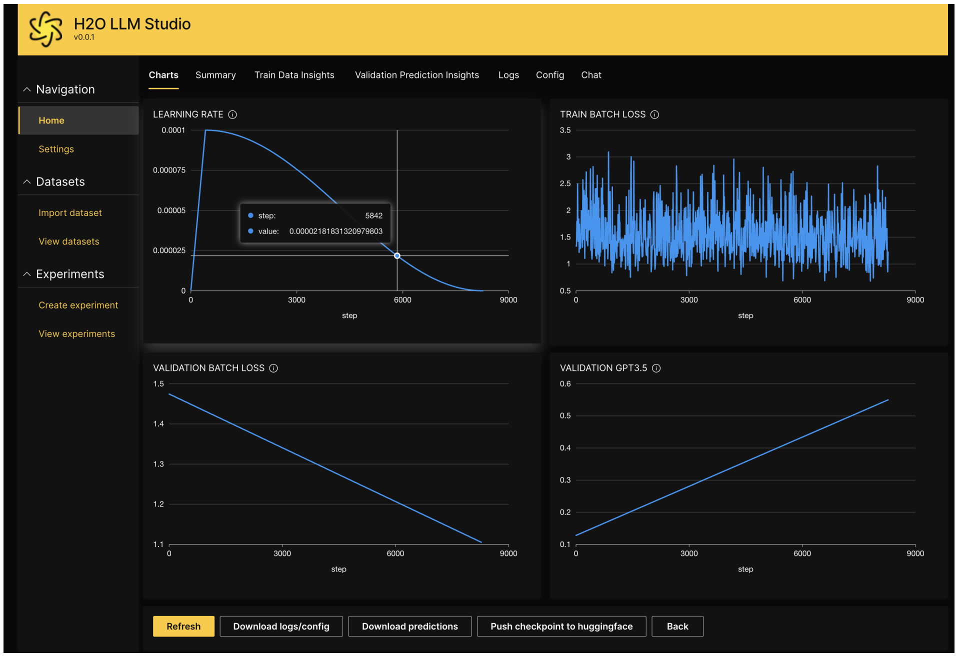
GPU #1 - current utilization: **85.0%** - VRAM usage: **59.0 GB** / **80.0 GB** - NVIDIA A100-SXM4-80GB

List of Datasets

name	problem type
oasst_v0	Text Causal Languag
oasst_v1	Text Causal Languag

List of Experiments

name	dataset	problem type	metric	val metric
flawless-tarantula.1.1.1.1	oasst_v0	Text Causal Language Modeling	GPT3.5	0.55
flawless-tarantula.1	oasst_v0	Text Causal Language Modeling	GPT3.5	0.0
flawless-tarantula	oasst_v0	Text Causal Language Modeling	GPT3.5	0.0
dynamic-sparrow	oasst_v0	Text Causal Language Modeling	GPT3.5	0.0



step: 5842
value: 0.00002181831320979803

H2O LLM Studio features include:

- easily and effectively fine-tune LLMs without the need for any coding experience
- use a graphic user interface (GUI) specially designed for large language models finetune any LLM using a large variety of hyperparameters
- use recent finetuning techniques such as Low-Rank Adaptation (LoRA) and 8-bit model training with a low memory footprint
- use advanced evaluation metrics to judge generated answers by the model
- track and compare your model performance visually. In addition, Neptune integration can be used.
- chat with your model and get instant feedback on your model performance
- easily export your model to the Hugging Face Hub and share it with the community

		BoolQ	PIQA	HellaSwag	WinoGrande	ARC-e	ARC-c	OBQA
GPT-3	175B	60.5	81.0	78.9	70.2	68.8	51.4	57.6
Gopher	280B	79.3	81.8	79.2	70.1	-	-	-
Chinchilla	70B	83.7	81.8	80.8	74.9	-	-	-
PaLM	62B	84.8	80.5	79.7	77.0	75.2	52.5	50.4
PaLM-cont	62B	83.9	81.4	80.6	77.0	-	-	-
PaLM	540B	**88.0**	82.3	83.4	**81.1**	76.6	53.0	53.4
LLaMa	7B	76.5	79.8	76.1	70.1	72.8	47.6	57.2
	13B	78.1	80.1	79.2	73.0	74.8	52.7	56.4
	33B	83.1	82.3	82.8	76.0	**80.0**	**57.8**	58.6
	65B	85.3	82.8	**84.2**	77.0	78.9	56.0	**60.2**
h2oGPT	6.9B	61.6	76.8	67.0	61.6	65.4	35.6	38.1
	12B	66.9	76.6	68.0	63.7	62.2	35.1	37.4
	20B	71.3	77.8	72.6	66.1	68.9	44.2	40.0
	40B	85.2	**83.3**	83.1	77.5	78.0	54.6	48.8

Table 3: **Zero-shot performance on Common Sense Reasoning tasks. Other scores from LLaMa paper**.

3.2 Validation, Limitations, and Capabilities

We are aware that open-source LLMs with fully permissive licenses are not as capable as certain closed-sourced offerings. As the open-source community continues to learn and improve, the available models will become better, and reach a point where they will be more and more suited for commercial applications.

3.2.1 Evaluation Metrics

We used the EleutherAI evaluation harness to confirm that our fine-tuned LLMs still exhibit the same basic capabilities as the foundation models. Table 3 shows a comparison of performance on several common-sense reasoning tasks. Note that error bars are on the order of +/- 1.

We also used ShareGPT prompts and evaluated the answers provided by h2oGPT by asking the OpenAssistant reward model or an advanced LLM like GPT-3.5/4 for a score between 0 and 1, or for which of two answers is better. More details can be found on our GitHub repositories.

3.2.2 Current Weaknesses

h2oGPT fine-tuned LLMs exhibit the same biases and limitations as their underlying foundation models, including:

- Factual correctness
- Code completion
- Reasoning, chain-of-thought
- Mathematics and logic

3.2.3 Current Capabilities

h2oGPT fine-tuned LLMs exhibit certain capabilities that can exceed their underlying foundation models without requiring significant prompt engineering:

- General Chat
- Summarization
- Creativity
- Rephrasing
- Private Document Chat with fact-based answers (thanks to VectorDB integration)

4 Outlook

There are several roadmap items we intend to work on in the near future, but these might change based on customer/community feedback or new developments:

- Reinforcement Learning with Human Feedback in H2O LLM Studio
- Improved VectorDB document search using metadata, large-context, prompt-to-code generation
- Wizard LM for automatic high-quality data preparation
- Self-alignment (research)
- Use the latest available open-source models and techniques for architectural or data-specific improvements

5 Conclusion

We are excited to announce that we have open-sourced a range of essential code components that are instrumental in effectively fine-tuning Language Models (LLMs) and transforming them into advanced ChatBots and Document Search engines. Our commitment to open-source principles means that we provide 100% permissive access to data, models, and code, empowering the wider community to leverage and build upon our advancements.

Through our extensive research and development efforts, we have achieved the cutting-edge in data preparation and fine-tuning techniques for LLMs. The resulting models represent the state of the art in the field, while adhering to commercially viable licenses. We remain dedicated to maintaining our position at the forefront of the learning curve, continuously pushing the boundaries of what is achievable.

It's important to note that our existing products, such as H2O Driverless AI, H2O Hydrogen Torch, and H2O Document AI, have already incorporated LLMs and other deep learning models for several years. By harnessing the power of the GPT revolution, we ensure that all our products continue to benefit from the ongoing innovations in this rapidly evolving field.

We are excited to contribute to the advancement of the NLP community and look forward to the collective progress that will be accelerated by the availability of our open-sourced code and models.

References

This is partial list of references that we collected during the creation of h2oGPT. We'd like to thank all collaborators and open-source community members.

h2oGPT repositories and discussion channels

- h2oGPT `https://github.com/h2oai/h2ogpt`
- H2O LLM Studio `https://github.com/h2oai/h2o-llmstudio`
- H2O.ai on Hugging Face `https://huggingface.co/h2oai`
- H2O.ai Generative Discord Channel

LLM related code directly used for h2oGPT:

- Alpaca LoRa
- LoRa
- Hugging Face Transformers
- Hugging Face Datasets
- Hugging Face PEFT
- bitsandbytes
- PyTorch
- AutoGPTQ

Code to consider including:

- flan-alpaca
- text-generation-webui
- minimal-llama
- finetune GPT-NeoX
- GPTQ for LLaMa
- OpenChatKit on multi-GPU
- Non-Causal LLM
- OpenChatKit Offload
- Flan-alpaca

Some open source models:

- GPT-NeoXT-Chat-Base-20B
- GPT-NeoX
- GPT-NeoX-20B
- Pythia-6.9B
- Pythia-12B
- Flan-T5-XXL
- GPT-J-Moderation-6B
- OIG safety models
- BigScience-mT0
- BigScience-XP3
- BigScience-Bloomz

Some creative commons models that would be interesting to use:

- Galactica-120B
- LLaMa-small-pt
- LLaMa-64b-4bit

Papers/Repos

- Self-improve
- Coding
- self-reflection
- RLHF
- DERA
- HAI Index Report 2023
- LLaMa
- GLM-130B
- RWKV RNN
- Toolformer
- GPTQ
- Retro
- Clinical outperforms
- Chain-Of-Thought
- scaling law1
- Big-bench
- Natural-Instructions

Other projects:

- StackLLaMa
- Alpaca-CoT
- ColossalAIChat
- EasyLM
- Koala
- Vicuna
- Flan-Alpaca
- FastChat
- alpaca.http
- chatgpt-retrieval-plugin
- subtl.ai docs search on private docs
- gretel
- alpaca lora 4bit
- alpaca lora 4bit readme
- code alpaca
- serge
- BlinkDL
- MosaicCM
- OpenAI Plugins

- GPT3.5-Turbo-PGVector
- LLaMa-Adapter
- llama-index
- minimal-llama
- llama.cpp
- mmap
- lamma.cpp more
- TargetedSummarization
- OpenFlamingo
- Auto-GPT
- PrivateGPT

Apache2/etc. Data

- OIG 43M instructions (direct HF link)
- More on OIG
- DataSet Viewer
- Anthropic RLHF
- WebGPT_Comparisons
- Self_instruct
- 20BChatModelData

Apache2/MIT/BSD-3 Summarization Data

- xsum for Summarization
- Apache2 Summarization
- MIT summarization
- BSD-3 summarization
- OpenRail
- Summarize_from_feedback

Ambiguous License Data

- GPT-4-LLM
- GPT4All
- LinkGPT4
- ShareGPT52K
- ShareGPT_Vicuna
- ChatLogs
- Alpaca-CoT
- LaMini-LM

Non-commercial Data

- GPT-3 based Alpaca Cleaned
- Dolly

Prompt Engineering

- PEFT Prompt/P-tuning
- Prompt/P-tuning Nemo/NVIDIA
- Info
- Info2
- Prompt-Tuning
- P-tuning v2
- babyagi

Validation

- Bleu/Rouge/Meteor/Bert-Score
- LM Evaluation Harness

Generate Hyperparameters

- hot-to-generate
- Notes_on_Transformers Chpt5
- Notes_on_Transformers_Chpt10

Embeddings

- OpenAI Expensive?
- Leaderboard

Commercial products

- OpenAI
- OpenAI Tokenizer
- OpenAI Playground
- OpenAI Chat
- OpenAI GPT-4 Chat
- cohere
- coherefinetune
- DocsBotAI
- Perplexity
- VoiceFlow
- NLPCloud

Inference

- FasterTransformer
- Kubernetes Triton
- Optimum
- MLC-LLM
- Triton Inference server

Semi-Open source Semi-Commercial products

- OpenAssistant
- OpenAssistant Repo
- OpenChatKit
- OpenDataHub
- OpenChatKit3
- OpenChatKit4
- langchain
- langchain+pinecone

Q/A docs

- HUMATA
- OSSCHat
- NeuralSearchCohere
- ue5

AutoGPT type projects

- AgentGPT
- Self-DEBUG
- BabyAGI
- AutoPR

Cloud fine-tune

- AWS
- AWS2

Chatbots

- GPT4ALL Chat
- GLT4ALL
- OASSST
- FastChat
- Dolly
- HF Instructions
- DeepSpeed Chat
- LoraChat
- Tabby
- TalkToModel

LangChain related

- Gradio Tools
- LLM Agents
- Meta Prompt

Summaries

- LLMs

Hallucinations

- On the Dangers of Stochastic Parrots

6 Disclaimer

Please read this disclaimer carefully before using the large language model provided by h2oGPT. Your use of the model signifies your agreement to the following terms and conditions.

Biases and Offensiveness: The large language model is trained on a diverse range of internet text data, which may contain biased, racist, offensive, or otherwise inappropriate content. By using this model, you acknowledge and accept that the generated content may sometimes exhibit biases or produce content that is offensive or inappropriate. The developers of this repository do not endorse, support, or promote any such content or viewpoints.

Limitations: The large language model is an AI-based tool and not a human. It may produce incorrect, nonsensical, or irrelevant responses. It is the user's responsibility to critically evaluate the generated content and use it at their discretion.

Use at Your Own Risk: Users of this large language model must assume full responsibility for any consequences that may arise from their use of the tool. The developers and contributors of this repository shall not be held liable for any damages, losses, or harm resulting from the use or misuse of the provided model.

Ethical Considerations: Users are encouraged to use the large language model responsibly and ethically. By using this model, you agree not to use it for purposes that promote hate speech, discrimination, harassment, or any form of illegal or harmful activities.

Reporting Issues: If you encounter any biased, offensive, or otherwise inappropriate content generated by the large language model, please report it to the repository maintainers through the provided channels. Your feedback will help improve the model and mitigate potential issues.

Changes to this Disclaimer: The developers of this repository reserve the right to modify or update this disclaimer at any time without prior notice. It is the user's responsibility to periodically review the disclaimer to stay informed about any changes.

By using the large language model provided in this repository, you agree to accept and comply with the terms and conditions outlined in this disclaimer. If you do not agree with any part of this disclaimer, you should refrain from using the model and any content generated by it.

Online version: Disclaimer