香港中文大學(深圳)
The Chinese University of Hong Kong, Shenzhen

DDA6307/CSC6052/MDS6002: Natural Language Processing
自然语言处理

Lecture 12: Tips for the Final Project

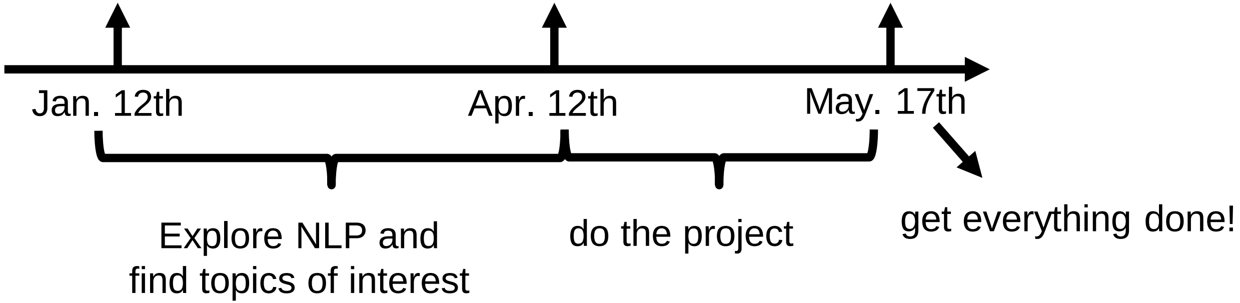
Spring 2024

Benyou Wang
School of Data Science

Final project

Timeline:

- **Poster Presentation:** May. 17th.
    - Prepare and present a poster showcasing your project's idea and progress to peers.
- **Final Project Deadline:** May. 17th (final date).
    - Submit a detailed final report elucidating the work done on the project.



TA supervision

Timeline:

April 26th: 14:00 pm to 16:30 pm (all TAs attending, we could make any possible adjustment for TA supervison)

May 6th:

- 9:00 - 10:30: TA Xidong WANG Group (Daoyuan 501)
- 10:30 - 12:00: TA Juhao LIANG Group (Daoyuan 501)
- 13:30 - 15:00: TA Fei YU Group (Daoyuan 501)
- 15:00 - 16:30: TA Junying CHEN Group (Daoyuan 501)

May 10th:

- 9:00 - 9:40: TA Xidong WANG Group (Daoyuan 501)
- 9:40 - 10:20: TA Juhao LIANG Group
- 10:20 - 11:00: TA Fei YU Group
- 11:00 - 11:40: TA Junying CHEN Group
- 13:30 - 16:20: Instructor Time (location: TB202)

Practice

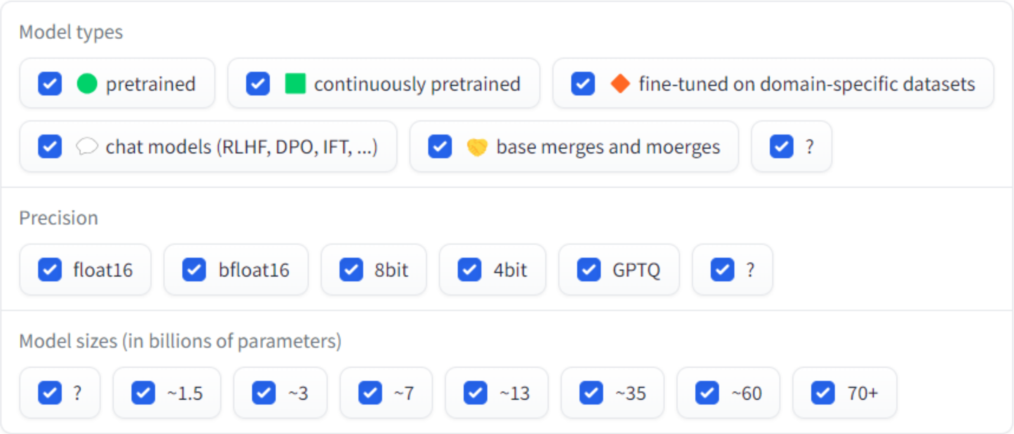
- Backbone model selection
- Dataset
- Model inference
    - vLLM
    - 百度千帆
- Model training
    - Pretraining (not suggested)
    - Supervised finetuning
    - Reinforced Learing fron Human Feedback (e.g., DPO)
    - RAG (Langchain or LLamaindex)
    - Agent (React)
- UI (Gradio)

https://bizon-tech.com/gpu-benchmarks/NVIDIA-Tesla-V100-vs-NVIDIA-RTX-3090-vs-NVIDIA-RTX-3060-vs-NVIDIA-A100-80-GB-(PCIe)/535vs579vs590vs624

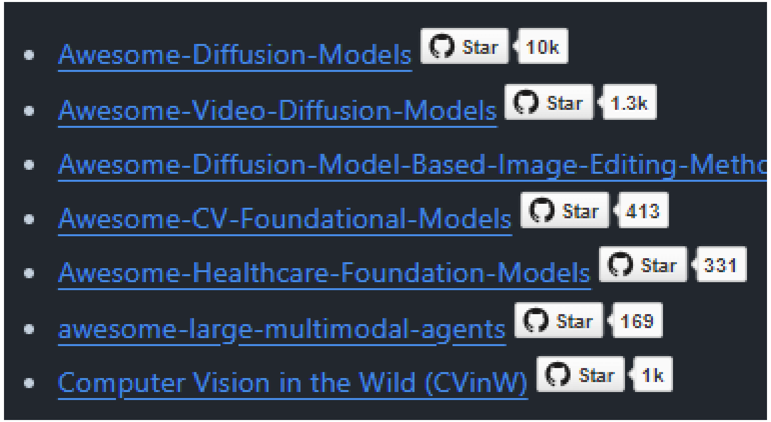
Models Selection

Refer to *LeaderBoard* and *Gihub Repo*

Open LLM Leaderboard *Filter LLMs here*



Awesome-Foundation-Models



- ❖ Precision of LLMs: The level of detail that a LLM uses when it processes numbers. Lower precision uses fewer bits, resulting in faster but less detailed computations.

- ❖ Where to Find LeaderBoard: Huggingface Space and PaperWithCode
- ❖ Also refer to Github repo with *Owesome*

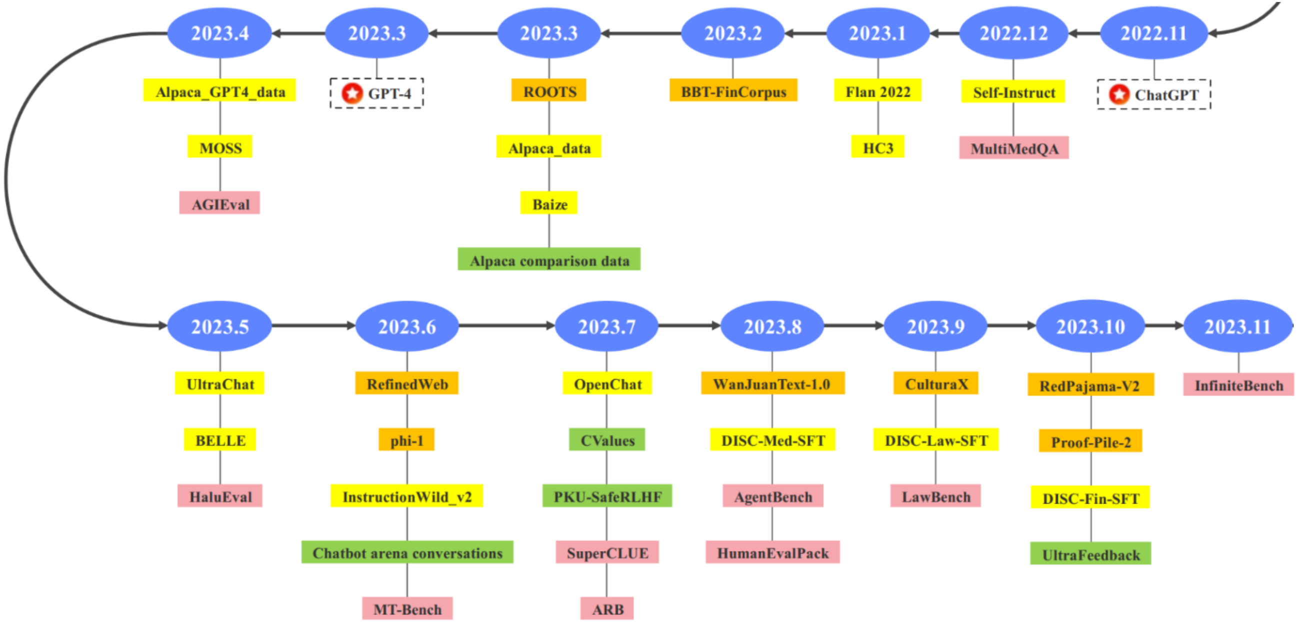
Awesome Tiny Models

- LLaMA 3
  - LLaMA 3 8B (not that small) https://huggingface.co/meta-llama/Meta-Llama-3-8B
- Qwen
  - Qwen1.5-0.5B: https://huggingface.co/Qwen/Qwen1.5-0.5B
  - Qwen-1_8B: https://huggingface.co/Qwen/Qwen-1_8B
  - Qwen1.5-1.8B: https://huggingface.co/Qwen/Qwen1.5-1.8B
- MiniCPM
  - MiniCPM-2B-128k: https://huggingface.co/openbmb/MiniCPM-2B-128k
- Phi
  - Phi-3-mini-128k-instruct: https://huggingface.co/microsoft/Phi-3-mini-128k-instruct
  - phi-2: https://huggingface.co/microsoft/phi-2
- Gemma
  - gemma-2b: https://huggingface.co/google/gemma-2b
  - gemma-1.1-2b-it: https://huggingface.co/google/gemma-1.1-2b-it

- Medical
  - Apollo-0.5B: https://huggingface.co/FreedomIntelligence/Apollo-0.5B
  - Apollo-1.8B: https://huggingface.co/FreedomIntelligence/Apollo-1.8B
- Multimodal model:
  - ALLaVA-3B: https://huggingface.co/FreedomIntelligence/ALLaVA-3B
  - MiniCPM-V-2: https://huggingface.co/openbmb/MiniCPM-V-2

Datasets

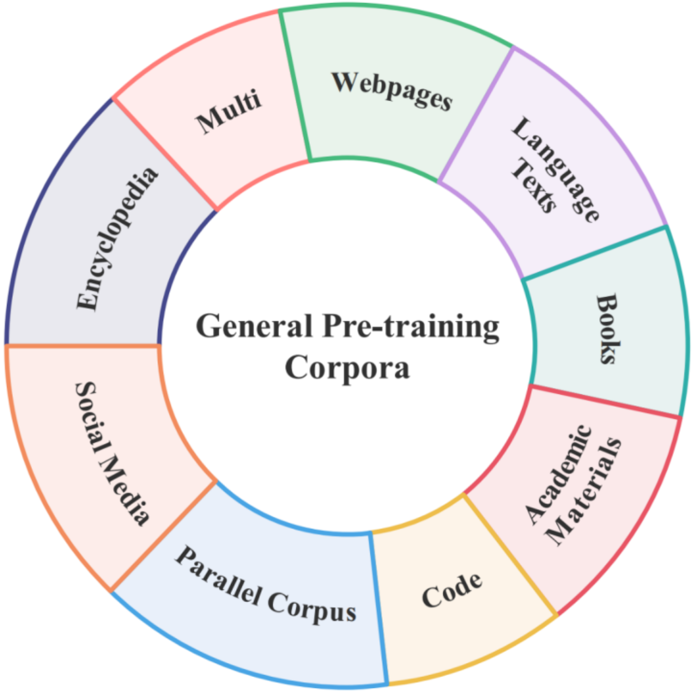
Datasets in ChatGPT Era



A timeline of some representative LLM datasets. **Orange** represents pre-training corpora, **yellow** represents instruction fine-tuning datasets, **green** represents preference datasets, and **pink** represents evaluation datasets.

Liu Y, Cao J, Liu C, et al. Datasets for Large Language Models: A Comprehensive Survey[J]. arXiv preprint arXiv:2402.18041, 2024.

Data categories of Pre-training Corpora



Pre-training corpora can generally be categorized into these eight types.

Liu Y, Cao J, Liu C, et al. Datasets for Large Language Models: A Comprehensive Survey[J]. arXiv preprint arXiv:2402.18041, 2024.

Pre-training Corpora

1. Webpages

- **Common Crawl 2007-X**
  - Publisher: Common Crawl
  - Size: -
  - License: Common Crawl Terms of Use
  - Source: Web crawler data

- **RedPajama-V2 2023-10**
  - Publisher: Together Computer
  - Size: 30.4 T Tokens
  - License: Common Crawl Terms of Use
  - Source: Common Crawl, C4, etc.

- **C4 2019-10**
  - Publisher: Google Research
  - Size: 12.68 TB
  - License: Common Crawl Terms of Use
  - Source: Common Crawl

- **WanJuan-CC 2024-2**
  - Publisher: Shanghai Artifcial Intelligence Laboratory
  - Size: 1 T Tokens
  - License: CC-BY-4.0
  - Source: Common Crawl

1. https://github.com/togethercomputer/RedPajama-Data
2. https://commoncrawl.org/
3. https://huggingface.co/datasets/allenai/c4
4. https://opendatalab.org.cn/OpenDataLab/WanJuanCC

Pre-training Corpora

2. Books

- **the_pile_books3**
  - Publisher: EleutherAI
  - Size: 100.9 Gib
  - License: MiT
  - Source: Toronto Book Corpus

- **Anna's Archive**
  - Publisher: Anna
  - Size: 586.3 TB
  - License: -
  - Source: Sci-Hub, Library Genesis, etc.

3. Encyclopedia

- **Baidu baike**
  - Publisher: Baidu
  - Size: -
  - License: Baidu baike User Agreement
  - Source: Encyclopedic content data

- **Wikipedia**
  - Publisher: Wikimedia Foundation
  - Size: -
  - License: CC-BY-SA-3.0 & GFDL
  - Source: Encyclopedic content data

1. https://huggingface.co/datasets/the_pile_books3
2. https://annas-archive.org/datasets
3. https://baike.baidu.com/
4. https://huggingface.co/datasets/wikipedia

Pre-training Corpora

4. Academic Materials

- **arXiv**
  - Publisher: Paul Ginsparg et al.
  - Size: -
  - License: Terms of Use for arXiv APIs
  - Source: arXiv preprint
- **PubMed Central**
  - Publisher: NCBI
  - Size: -
  - License: PMC Copyright Notice
  - Source: Biomedical scientific literature
  - Category: Academic Materials

5. Code

- **Github**
  - Publisher: Microsoft
  - Size: -
  - License: -
  - Source: Various code projects
- **BIGQUERY**
  - Publisher: Salesforce Research
  - Size: 341.1 GB
  - License: Apache-2.0
  - Source: BigQuery

1. https://arxiv.org/
2. https://www.ncbi.nlm.nih.gov/pmc/
3. https://github.com/
4. https://github.com/salesforce/CodeGen

Pre-training Corpora

6. Language Texts

- **News-crawl**
  - Publisher: UKRI et al.
  - Size: 110 GB
  - License: CC0
  - Source: Newspapers

7. Parallel Corpus

- **WikiMatrix**
  - Publisher: Facebook AI et al.
  - Size: 134 M parallel sentences
  - License: CC-BY-SA
  - Source: Wikipedia

8. Language Texts

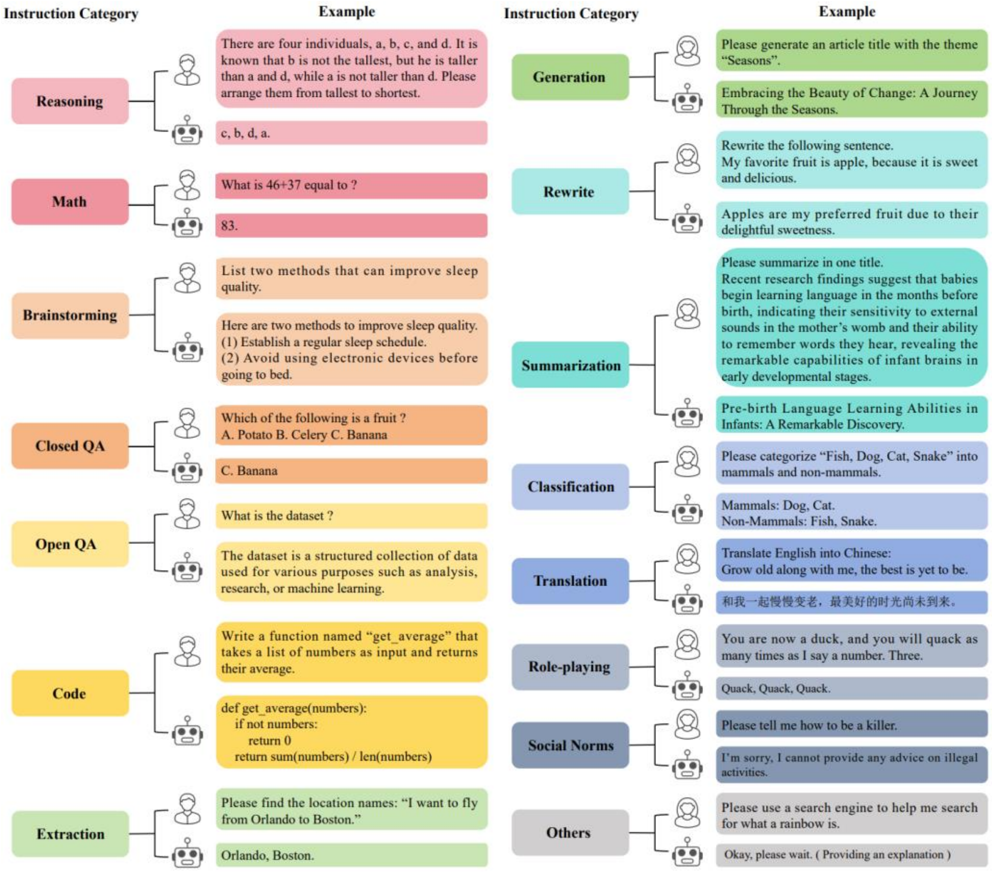
- **OpenWebText**
  - Publisher: Brown University
  - Size: 38 GB
  - License: CC0
  - Source: Reddit

1. https://data.statmt.org/news-crawl/
2. https://github.com/facebookresearch/LASER/tree/main/tasks/WikiMatrix
3. https://skylion007.github.io/OpenWebTextCorpus/

Instruction Fine-tuning Datasets

Fine-tuning datasets for AI models include text pairs of "instruction inputs" and "answer outputs." The **instruction inputs** are diverse requests like classification, summarization, and paraphrasing made by humans. The **answer outputs** are the model-generated responses that meet human expectations.

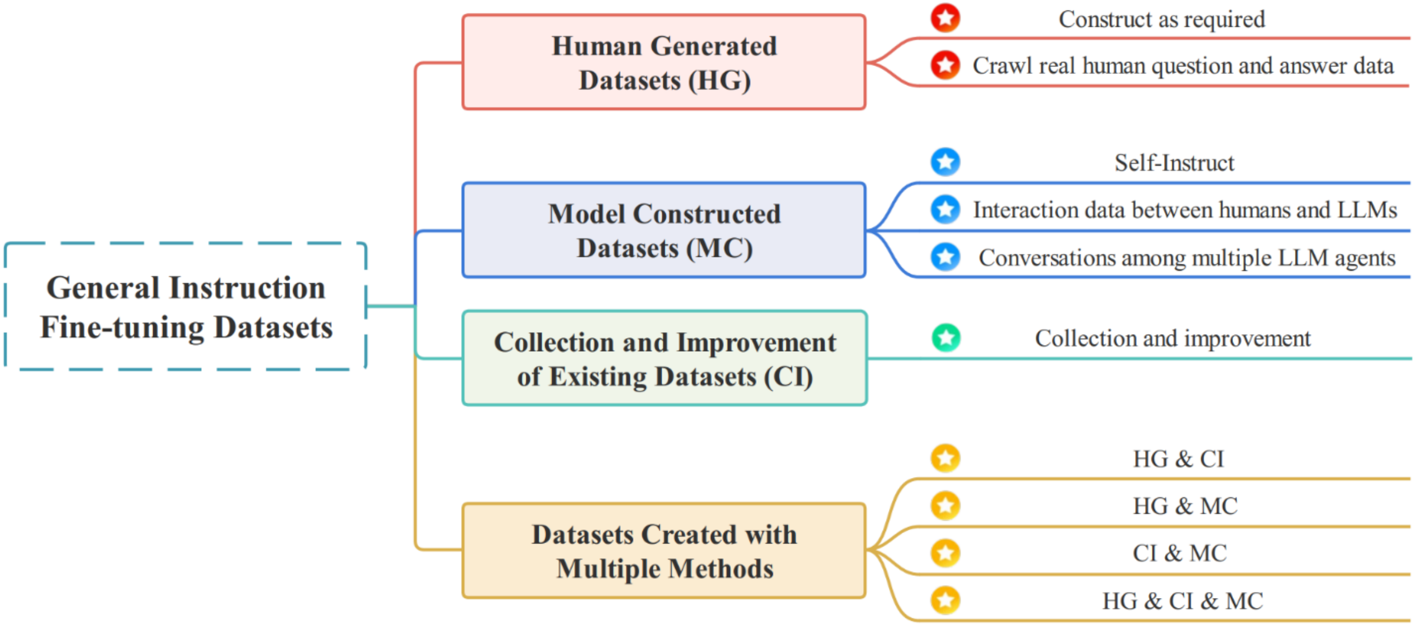
Summary of Instruction Categories



Considering the current classification status and focusing only on single-turn dialogue instructions, instructions are broadly grouped into these classes.

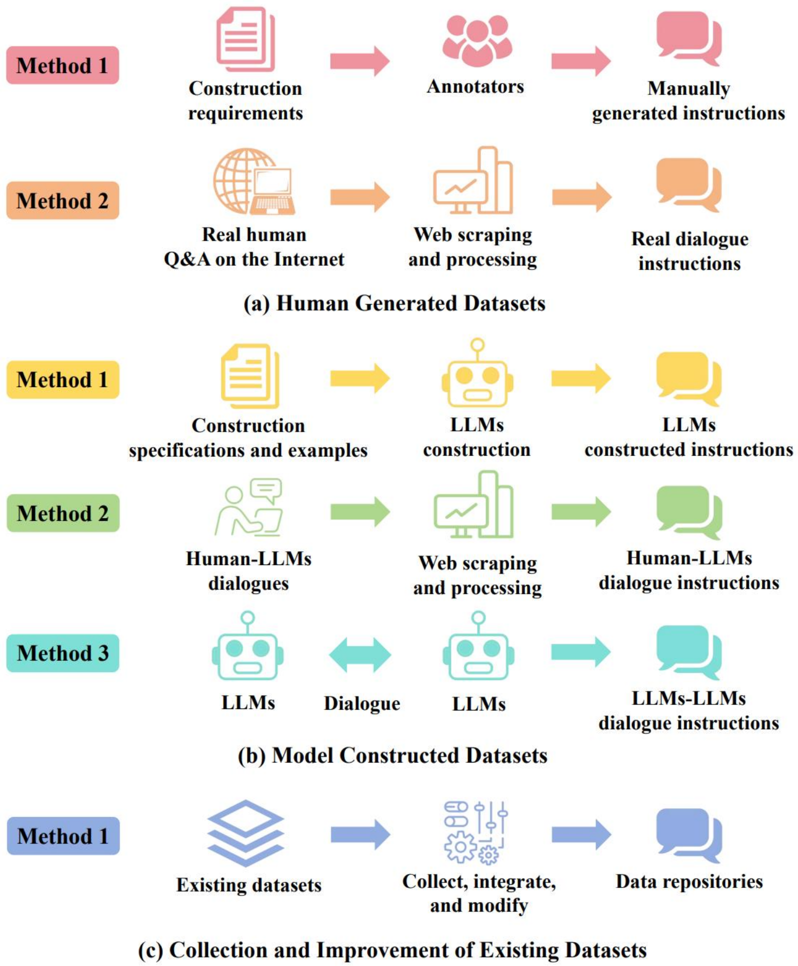
Liu Y, Cao J, Liu C, et al. Datasets for Large Language Models: A Comprehensive Survey[J]. arXiv preprint arXiv:2402.18041, 2024.

Instruction Fine-tuning Datasets



Construction methods corresponding to general instruction fine-tuning datasets

Instruction Fine-tuning Datasets



General Instruction Fine-tuning Datasets

- **Alpaca_GPT4**
  - Publisher: Microsoft Research
  - Size: 52K instances
  - License: Apache-2.0
  - Source: Generated by GPT-4 with Aplaca_data prompts

    (Low construction cost)

- **Wizard_evol_instruct_70K**
  - Publisher: Microsoft et al.
  - Size: 70K instances
  - License: -
  - Source: Evolve instructions through the Evol-Instruct method

    (High quality)

- **Phoenix-sft-data**
  - Publisher: **CUHKSZ**
  - Size: 46K instances
  - License: CC-BY-4.0
  - Source: Multi-lingual instructions, post-translated multi-lingual instructions

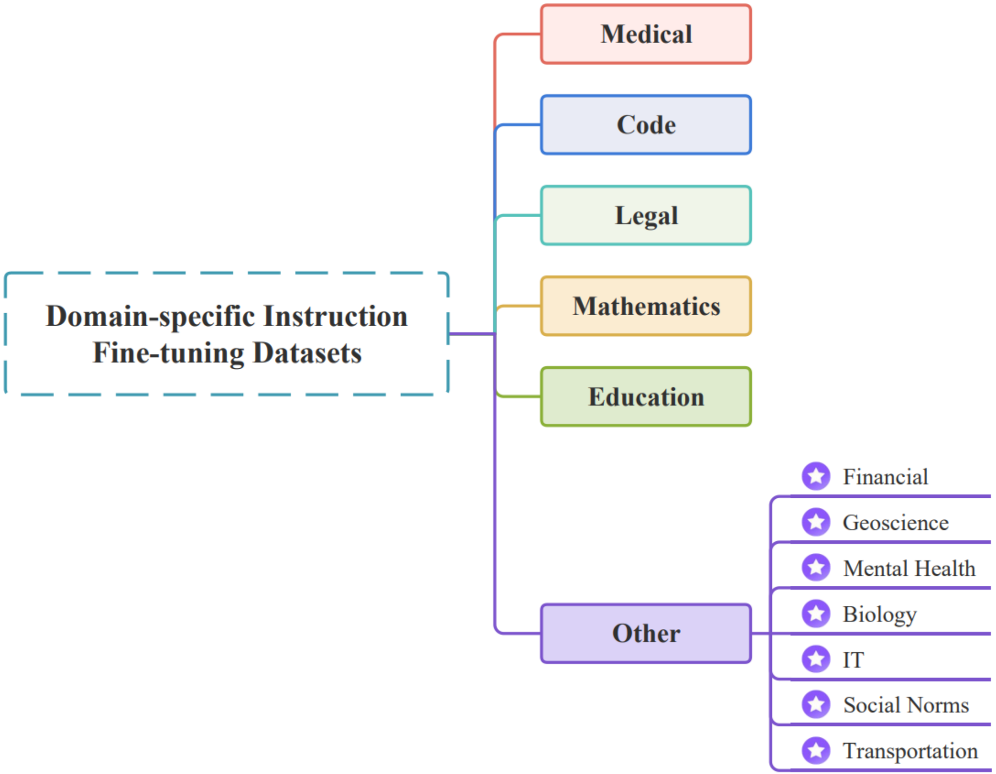
    (Multilingual)

- **ShareGPT90K**
  - Publisher: RyokoAI
  - Size: 90K instances
  - License: CC0
  - Source: ShareGPT

    (multi-round dialogue)

1. https://github.com/Instruction-Tuning-with-GPT-4/GPT-4-LLM#data-release 2. https://github.com/nlpxucan/WizardLM
3. https://github.com/FreedomIntelligence/LLMZoo 4. https://huggingface.co/datasets/RyokoAI/ShareGPT52K

Domain-specific Instruction Fine-tuning Datasets



Domain categories of the domain-specific instruction fine-tuning datasets

Domain-specific Instruction Fine-tuning Datasets

1. Medical

- **Huatuo-26M**
  - Publisher: **CUHKSZ**
  - Size: 26504088 instances
  - License: Apache-2.0
  - Source: Collection and improvement of various datasets.

- **HuatuoGPT**
  - Publisher: **CUHKSZ**
  - Size: 226042 instances
  - License: Apache-2.0
  - Source: Real conversations between doctors and patients & Generated by ChatGPT

- **HuatuoGPT-II**
  - Publisher: **CUHKSZ**
  - Size: 5394K instances
  - License: Apache-2.0
  - Source: Construct medical instructions from medical corpus.

The HuatuoGPT series has garnered significant attention, accumulating over 1,000 stars, more than 500,000 uses, and over 65 citations.

1. https://github.com/FreedomIntelligence/Huatuo-26M 2. https://github.com/FreedomIntelligence/HuatuoGPT
3. https://github.com/FreedomIntelligence/HuatuoGPT-II

Domain-specific Instruction Fine-tuning Datasets

2. Code

- **Code_Alpaca_20K**
  - Publisher: Sahil Chaudhary
  - Size: 20K instances
  - License: Apache-2.0
  - Source: Generated by Text-Davinci-003

- **CodeContest**
  - Publisher: DeepMind
  - Size: 13610 instances
  - License: Apache-2.0
  - Source: Collection and improvement of various datasets

3. Legal

- **DISC-Law-SFT**
  - Publisher: Fudan University et al.
  - Size: 403K instances
  - License: Apache-2.0
  - Source: Open source datasets & Legal-related Text Content

- **HanFei**
  - Publisher: CAS & **CUHKSZ**
  - Size: 255K instances
  - License: Apache-2.0
  - Source: Filter legal-related data according to rules

1. https://github.com/sahil280114/codealpaca 2. https://github.com/siat-nlp/HanFei
3. https://github.com/FudanDISC/DISC-LawLLM 4. https://github.com/salesforce/CodeGen

Domain-specific Instruction Fine-tuning Datasets

4. Math

- **OpenMathInstruct**
  - Publisher: NVIDIA
  - Size: 1.8M instances
  - License: NVIDIA License
  - Source: GSM8K and MATH datasets (original questions);
- **OVM**
  - Publisher: **CUHKSZ**
  - Size: 13610 instances
  - License: Apache-2.0
  - Source: The reasoning instruction of GSM8K and game24

5. Education

- **Child_chat_data**
  - Publisher: **CUHKSZ**
  - Size: 12000 instances
  - License: -
  - Source: storybooks and science books
- **Child_chat_data**
  - Publisher: Harbin Institute of Technology et al.
  - Size: 5000 instances
  - License: -
  - Source: Real conversations & Generated by GPT-3.5-Turbo

1. https://github.com/sahil280114/codealpaca 2. https://huggingface.co/datasets/FreedomIntelligence/OVM-dataset
3. https://github.com/HIT-SCIR-SC/QiaoBan 4. https://github.com/FreedomIntelligence/RichGPT

README

InstructionZoo

A collection of open-source Instruction-tuning dataset to train chat-based LLMs (ChatGPT,LLaMA,Alpaca).

This is an on-going project. We will soon add tags to classify the following datasets and continuously update our collection.

Table of Contents

Custom properties

255 stars

12 watching

23 forks

Report repository

Releases

No releases published

Create a new release

Packages

No packages published

Publish your first package

Contributors 4

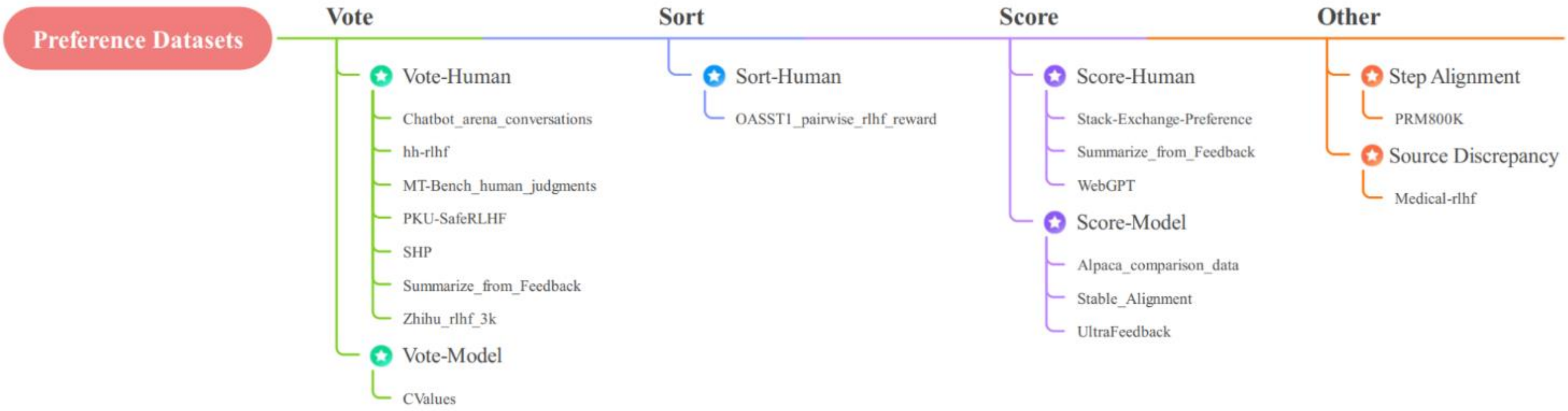
- zzhheloise Zhihan ZHANG
- sileod
- wabyking waby
- zhjohnchan Zhihong Chen

https://github.com/FreedomIntelligence/InstructionZoo

Preference Datasets

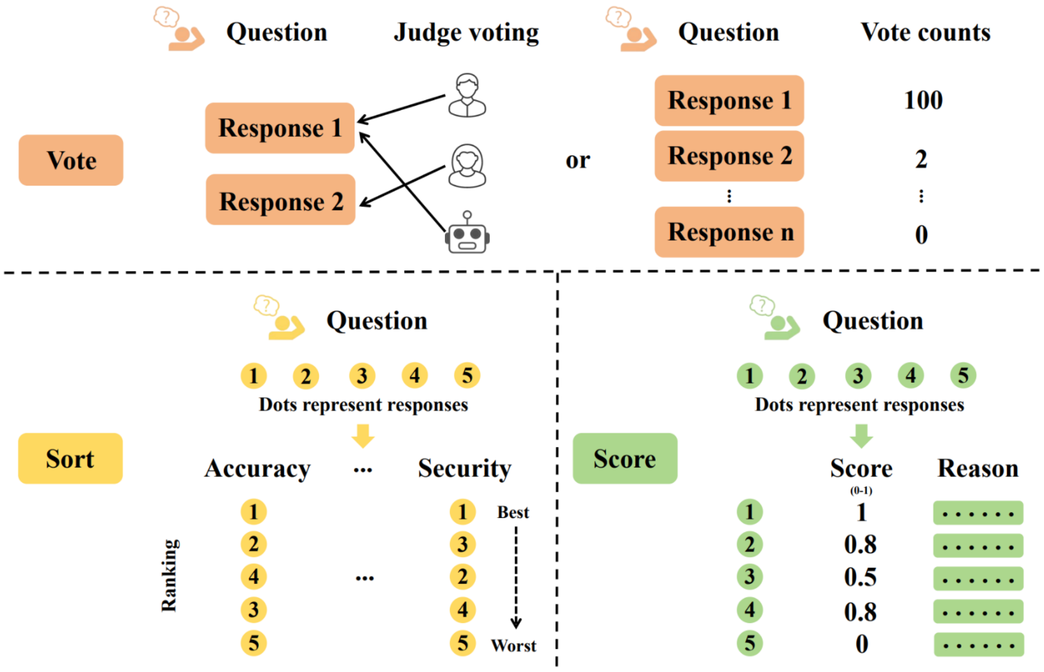
Preference datasets are compilations of instructional prompts that include evaluations of various responses to the same input. These datasets generally feature paired instructions, each accompanied by distinct responses, and are supplemented with feedback from either humans or alternative models.

Preference Datasets



Different preference datasets corresponding to various preference evaluation methods

Preference Datasets



Different preference evaluation methods

Liu Y, Cao J, Liu C, et al. Datasets for Large Language Models: A Comprehensive Survey[J]. arXiv preprint arXiv:2402.18041, 2024.

Preference Datasets

- **Chatbot_arena_conversations**
  - Publisher: UC Berkeley et al.
  - Size: 33000 instances
  - License: CC-BY-4.0 & CC-BY-NC-4.0
  - Preference: **Vote**
  - Source: Generated by twenty LLMs & Manual judgment

- **Stack-Exchange-Preferences**
  - Publisher: Anthropic
  - Size: 10807695 instances
  - License: CC-BY-SA-4.0
  - Preference: **Score**
  - Source: Stackexchange data & Manual scoring

- **OASST1_pairwise_rlhf_reward**
  - Publisher: Tasksource
  - Size: 18918 instances
  - License: Apache-2.0
  - Preference: **Sort**
  - Source: OASST1 datasets & Manual sorting

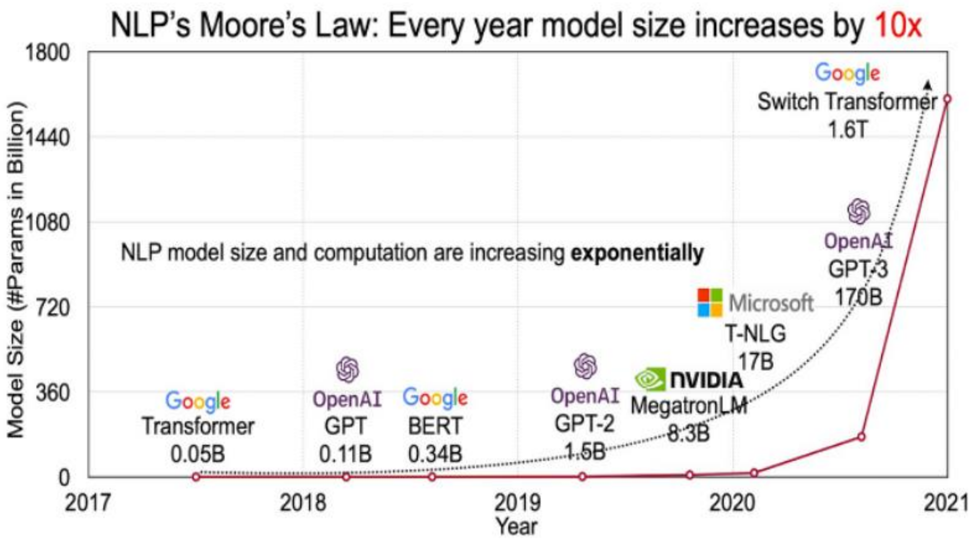
1. https://browse.arxiv.org/pdf/2306.05685.pdf 2. https://huggingface.co/datasets/tasksource/oasst1_pairwise_rlhf_reward
3. https://huggingface.co/datasets/HuggingFaceH4/stack-exchange-preferences

vLLM

vLLM - Background

Large language models (LLMs) are getting larger and larger, and the GPU memory overhead for training and deployment is also increasing!

Models (float32)	Inference Memory Estimation
Llama-3-8B	32 GB
Llama-3-70B	280 GB



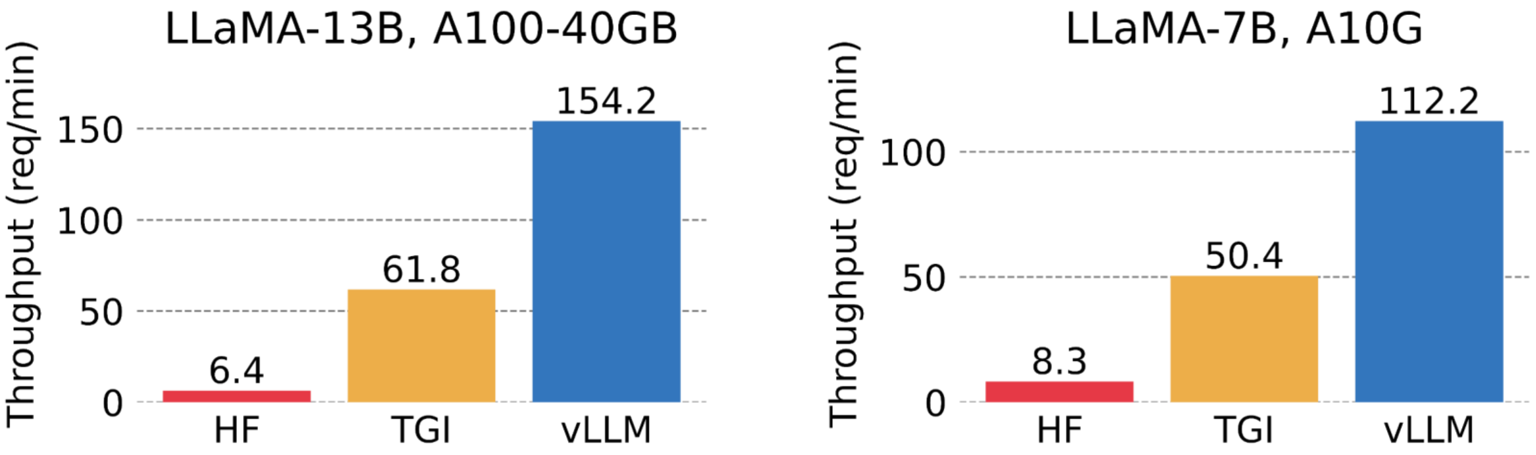
Memory

	NVIDIA RTX 3060	NVIDIA RTX 3090	NVIDIA Tesla V100	NVIDIA A100 80 GB (PCIe)
Bandwidth	360 GB/s	936.2 GB/s	897 GB/s	2039 GB/s
Memory Bus	192 bit	384 bit	4096 bit	5120 bit
Memory Size	12 GB	24 GB	32 GB	80 GB
Memory Type	GDDR6	GDDR6X	HBM2	HBM2e

https://bizon-tech.com/gpu-benchmarks/NVIDIA-Tesla-V100-vs-NVIDIA-RTX-3090-vs-NVIDIA-RTX-3060-vs-NVIDIA-A100-80-GB-(PCIe)/535vs579vs590vs624

vLLM

A **high-throughput** and **memory-efficient** inference and serving engine for LLMs

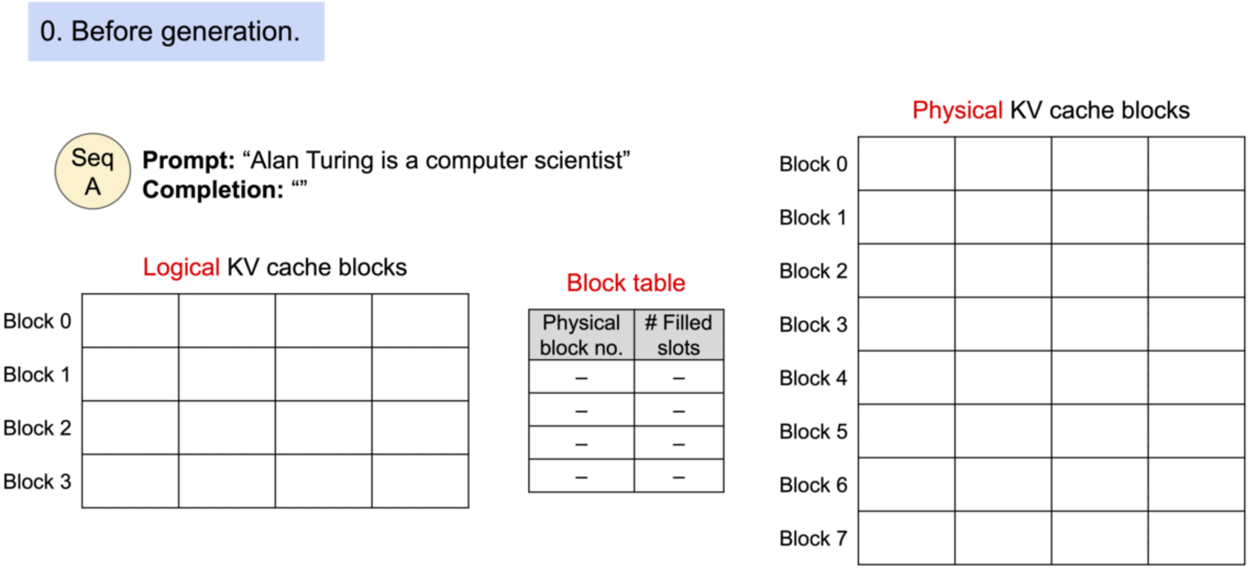


Serving throughput when each request asks for one output completion. vLLM achieves 14x - 24x higher throughput than HF and 2.2x - 2.5x higher throughput than TGI.

https://docs.vllm.ai/en/latest/

vLLM

PagedAttention, an attention algorithm inspired by the classic idea of virtual memory and paging in operating systems.



Example generation process for a request with ***PagedAttention***.

https://blog.vllm.ai/2023/06/20/vllm.html

Get started with vLLM

Install vLLM with the following command (check out our installation guide for more):

```
$ pip install vllm
```

vLLM can be used for both offline inference and online serving. To use vLLM for offline inference, you can import vLLM and use the `LLM` class in your Python scripts:

```
from vllm import LLM

prompts = ["Hello, my name is", "The capital of France is"]  # Sample prompts.
llm = LLM(model="lmsys/vicuna-7b-v1.3")  # Create an LLM.
outputs = llm.generate(prompts)  # Generate texts from the prompts.
```

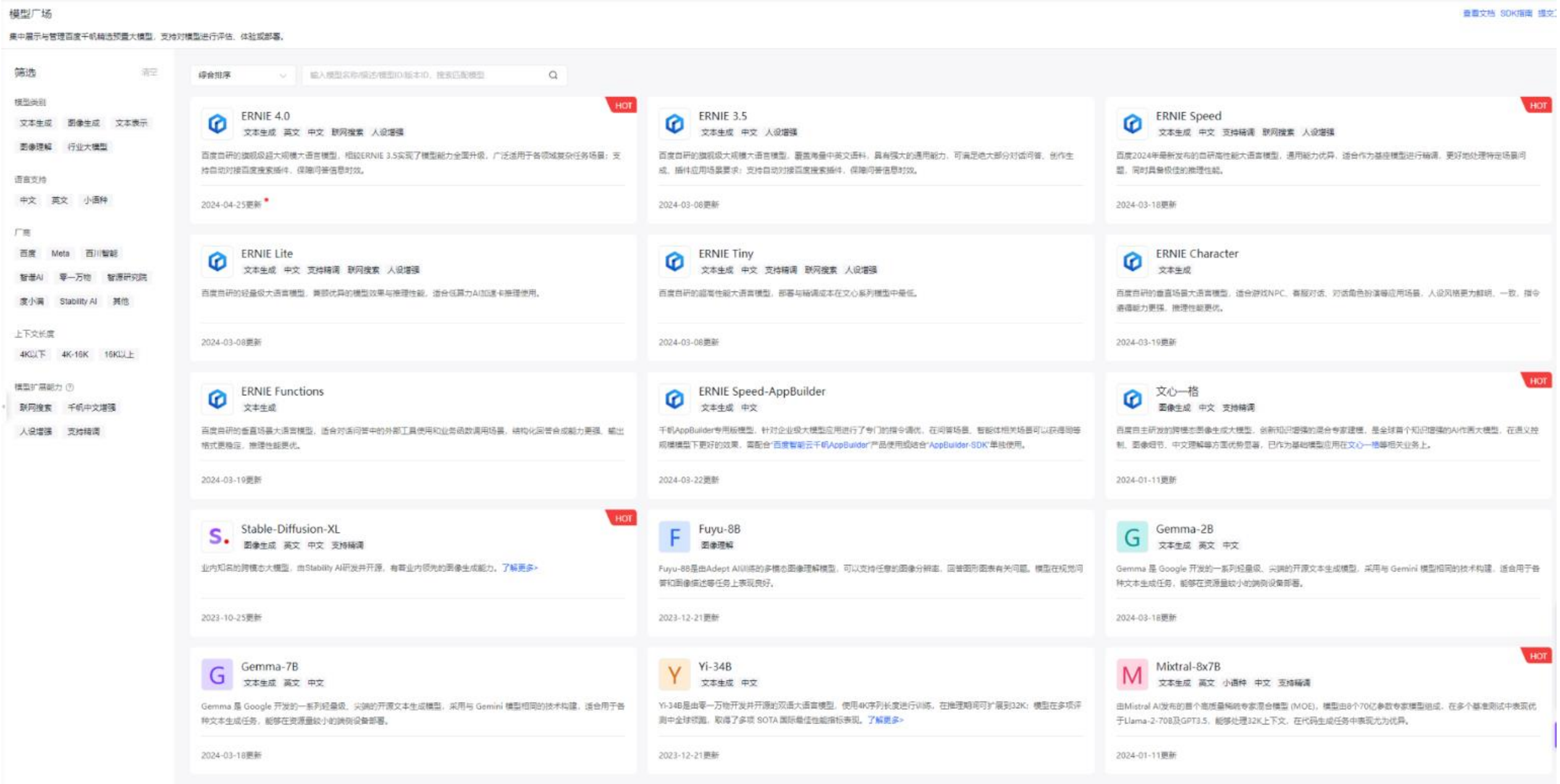
To use vLLM for online serving, you can start an OpenAI API-compatible server via:

```
$ python -m vllm.entrypoints.openai.api_server --model lmsys/vicuna-7b-v1.3
```

You can query the server with the same format as OpenAI API:

```
$ curl http://localhost:8000/v1/completions \
    -H "Content-Type: application/json" \
    -d '{
        "model": "lmsys/vicuna-7b-v1.3",
        "prompt": "San Francisco is a",
        "max_tokens": 7,
        "temperature": 0
    }'
```

https://docs.vllm.ai/en/latest/getting_started/quickstart.html



https://cloud.baidu.com/product/wenxinworkshop

Efficient Model Fine-tuning

PEFT: State-of-the-art Parameter-Efficient Fine-Tuning.

High performance on consumer hardware

Consider the memory requirements for training the following models on the dataset with an A100 80GB GPU with more than 64GB of CPU RAM.

Model	Full Finetuning	PEFT-LoRA PyTorch	PEFT-LoRA DeepSpeed with CPU Offloading
bigscience/T0_3B (3B params)	47.14GB GPU / 2.96GB CPU	14.4GB GPU / 2.96GB CPU	9.8GB GPU / 17.8GB CPU
bigscience/mt0-xxl (12B params)	OOM GPU	56GB GPU / 3GB CPU	22GB GPU / 52GB CPU
bigscience/bloomz-7b1 (7B params)	OOM GPU	32GB GPU / 3.8GB CPU	18.1GB GPU / 35GB CPU

With ***LoRA*** you can fully finetune a 12B parameter model that would've otherwise run out of memory on the 80GB GPU, and comfortably fit and train a 3B parameter model. When you look at the 3B parameter model's performance, it is comparable to a fully finetuned model at a fraction of the GPU memory.

Submission Name	Accuracy
Human baseline (crowdsourced)	0.897
Flan-T5	0.892
lora-t0-3b	0.863

https://github.com/huggingface/peft

PEFT: Quantization

Quantization is another method for reducing the memory requirements of a model by representing the data in a lower precision. It can be combined with PEFT methods to make it even easier to train and load LLMs for inference.

Load Pre-trained model and tokenizer

First let's load the model we are going to use - phoenix-inst-chat-7b! Note that the model itself is around 7B in full precision

```python
import torch
from transformers import AutoTokenizer, AutoModelForCausalLM, BitsAndBytesConfig

# Quantization type (fp4 or nf4), According to QLoRA paper, for training 4-bit base models (e.g. using LoRA adapters) one should use
bnb_4bit_quant_type = "nf4"

# Activate nested quantization for 4-bit base models (double quantization)
use_nested_quant = True

model_id = "FreedomIntelligence/phoenix-inst-chat-7b"
bnb_config = BitsAndBytesConfig(
    load_in_4bit=True,
    bnb_4bit_use_double_quant=use_nested_quant,
    bnb_4bit_quant_type=bnb_4bit_quant_type,
    bnb_4bit_compute_dtype=torch.bfloat16
)

tokenizer = AutoTokenizer.from_pretrained(model_id)
model = AutoModelForCausalLM.from_pretrained(model_id, quantization_config=bnb_config, device_map={"":0})
```

assignment 3 example

https://github.com/huggingface/peft

PEFT: Quickstart

Install PEFT from pip:

```
pip install peft
```

Prepare a model for training with a PEFT method such as LoRA by wrapping the base model and PEFT configuration with `get_peft_model`. For the bigscience/mt0-large model, you're only training 0.19% of the parameters!

```
from transformers import AutoModelForSeq2SeqLM
from peft import get_peft_config, get_peft_model, LoraConfig, TaskType
model_name_or_path = "bigscience/mt0-large"
tokenizer_name_or_path = "bigscience/mt0-large"

peft_config = LoraConfig(
    task_type=TaskType.SEQ_2_SEQ_LM, inference_mode=False, r=8, lora_alpha=32, lora_dropout=0.
)

model = AutoModelForSeq2SeqLM.from_pretrained(model_name_or_path)
model = get_peft_model(model, peft_config)
model.print_trainable_parameters()
"trainable params: 2359296 || all params: 1231940608 || trainable%: 0.19151053100118282"
```

https://huggingface.co/docs/peft/quicktour

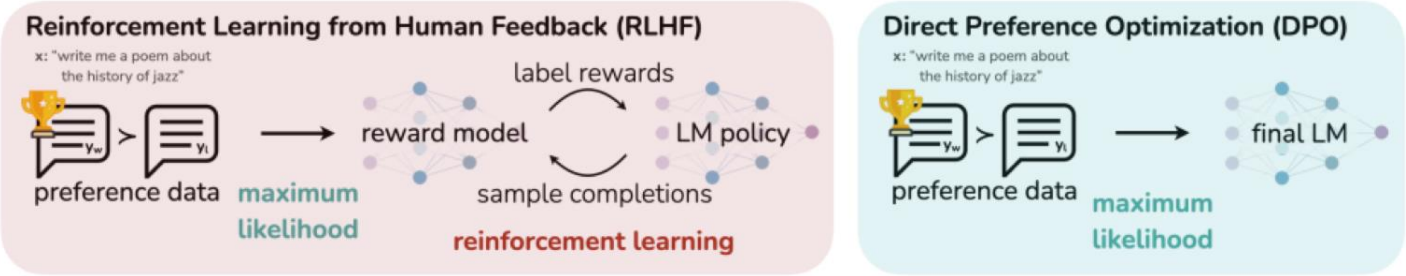
DPO
Direct Preference Optimization

DPO

Direct Preference Optimization (DPO) is a RLHF algorithm, which modifies the online training algorithm PPO into **offline training** via a new parameterization of the reward model

You can see the detailed derivation process in the paper "Direct Preference Optimization: Your Language Model is Secretly a Reward Model"

In DPO, a LLM is directly optimized on the preference data, without the need of reward model training



DPO

Preference data: (question, chosen, rejected), i.e. one question and two corresponding responses with indicating which one is preferred

- Example of preference data

Question	Chosen	Rejected
Human: What are pros and cons of electrical cars?	Assistant: Pro-electrical cars: 1) They charge using renewable energy 2) They are environmentally friendly, and require less energy than gasoline cars 3) They can use renewable energy sources like wind, and solar 4) They are quieter 5) They are more efficient 6) They are less polluting Con-electrical cars: 1) They have a limited range (if they need charging) 2) They are expensive, because they require batteries 3) They are a bit heavier than gasoline cars, due to the batteries	Assistant: The pros are they reduce pollution, and some also feel they're more fun to drive than gas-powered cars. The cons are that the batteries require electricity, which is a limited resource, and the cars are more expensive.

DPO

Popular open-source preference dataset:

- hh (helpfulness and harmlessness)

  released by Anthropic, the most classical one

  huggingface: Anthropic/hh-rlhf

- UltraFeedback

  released by THU, the latest large-scale preference dataset, very popular

  huggingface: openbmb/UltraFeedback

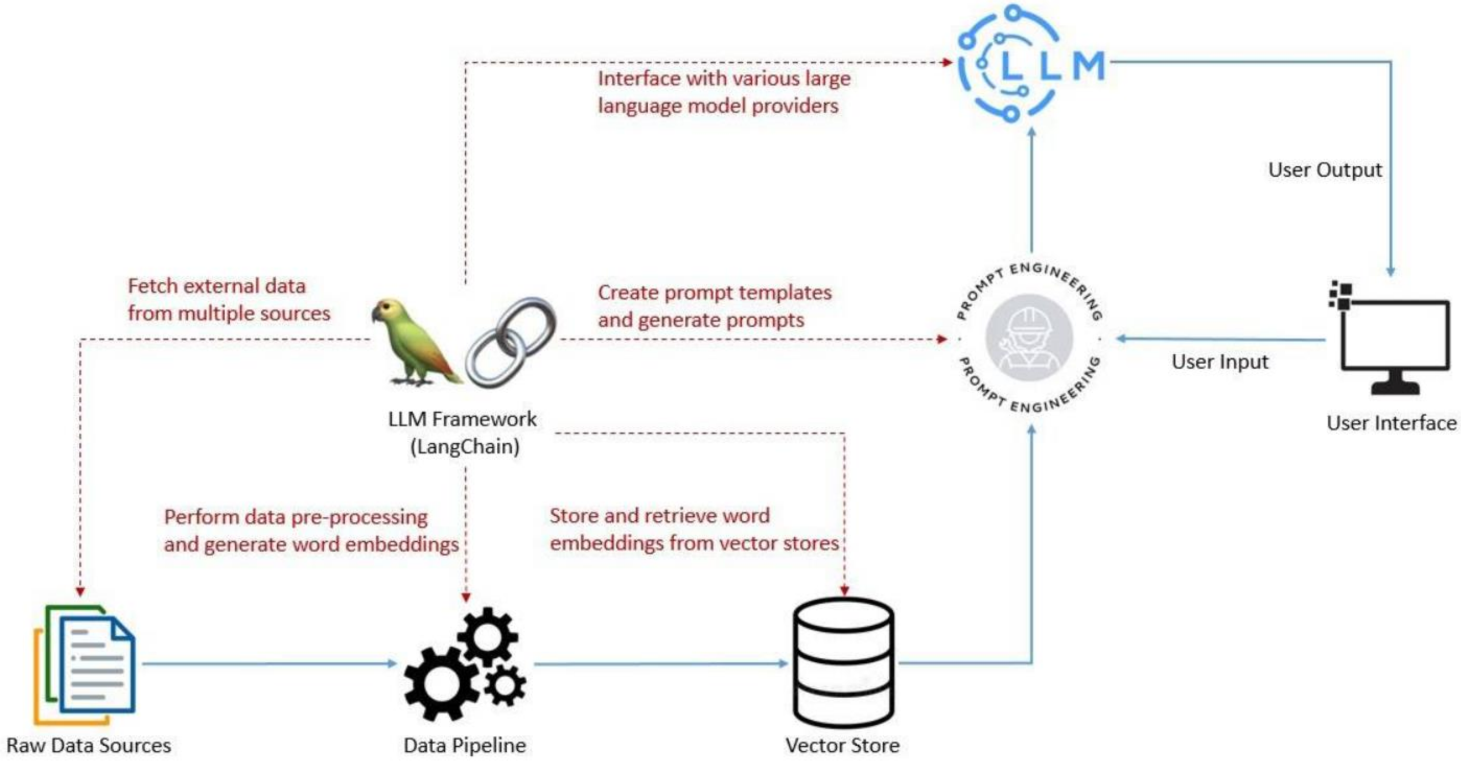
DPO

Implementation: based on DPOTrainer in trl, see code example

```
from trl import DPOTrainer

trainer = DPOTrainer(
    model,
    model_ref,
    tokenizer=tokenizer,
    train_dataset=train_dataset,
    eval_dataset=eval_dataset,
    beta=args.beta,
    loss_type=args.loss_type,
    args=training_args,
    max_length=args.max_length,
    max_prompt_length=args.max_prompt_length,
    max_target_length=args.max_target_length,
    generate_during_eval=args.generate_during_eval,
    disable_dropout=model_args.disable_dropout,
)
trainer.train()
trainer.save_model(training_args.output_dir)
```

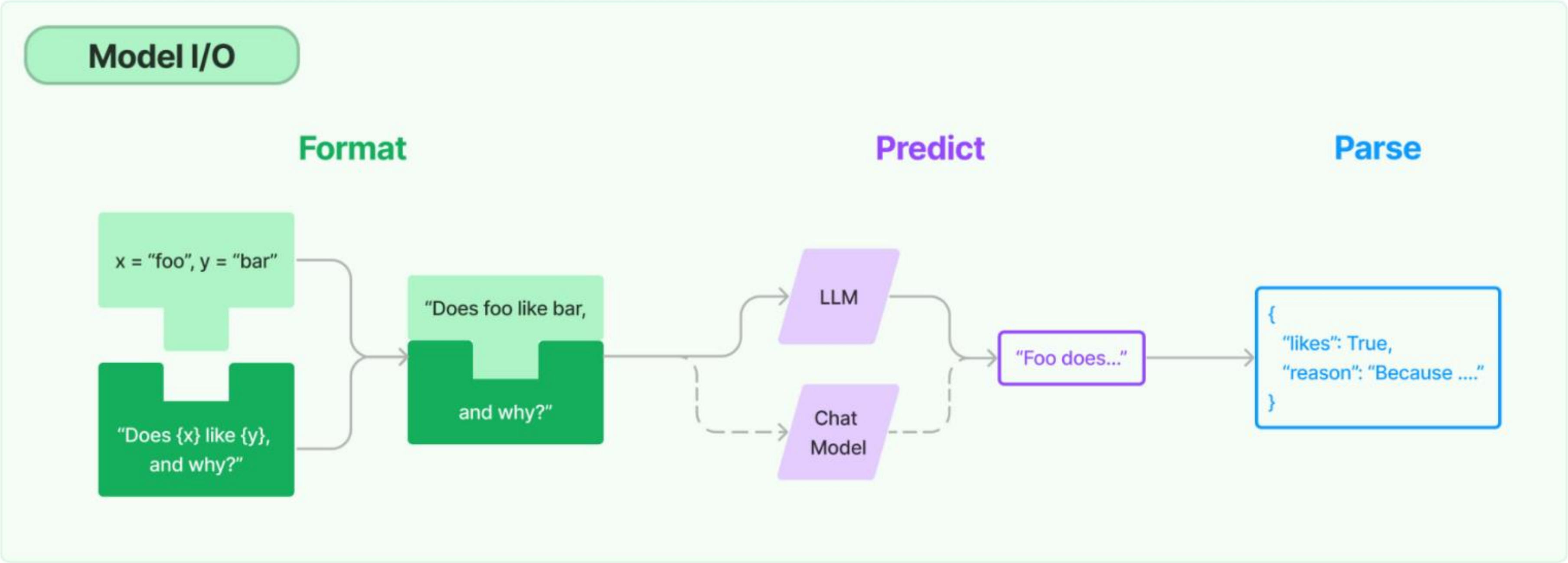
Langchain

What Is LangChain?



- An Framework Combining LLMs with external data.
- Multiple components to be called in a specific sequence.
  - This is what's referred to as a chain in LangChain.

Building Blocks of LangChain — Models I/O



- ❑ Templatize prompts
- ❑ Dynamically select and manage model inputs.
- ❑ Extract information from model outputs

Building Blocks of LangChain — Models I/O

```python
response = llm.invoke("List the seven wonders of the world.")
print(response)
```

```
1. Great Pyramid of Giza (Egypt)
2. Hanging Gardens of Babylon (Iraq)
3. Statue of Zeus at Olympia (Greece)
4. Temple of Artemis at Ephesus (Turkey)
5. Mausoleum at Halicarnassus (Turkey)
6. Colossus of Rhodes (Greece)
7. Lighthouse of Alexandria (Egypt)
```

```python
from langchain.llms import OpenAI
llm = OpenAI()
```

```python
from langchain.schema.messages import HumanMessage, SystemMessage
messages = [
    SystemMessage(content="You are Micheal Jordan."),
    HumanMessage(content="Which shoe manufacturer are you associated with?"),
]
response = chat.invoke(messages)
print(response.content)
```

```
I am associated with the Nike brand.
```

```python
from langchain.chat_models import ChatOpenAI
chat = ChatOpenAI()
```

- LLMs accept **strings** as inputs, or objects which can be coerced to string prompts

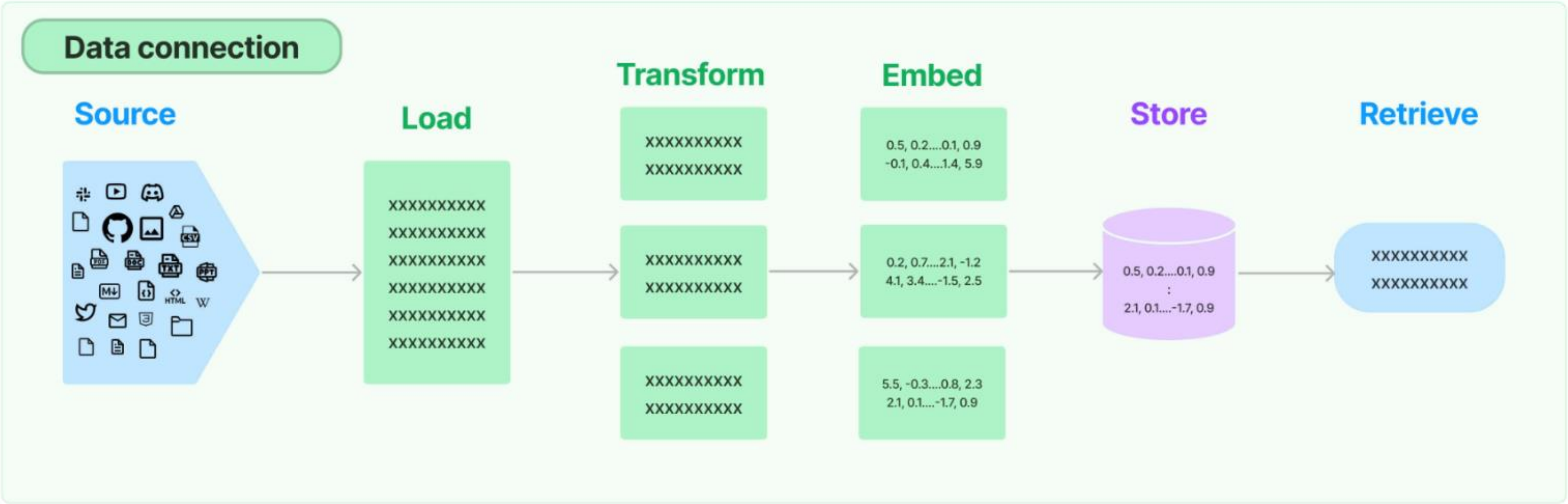
Building Blocks of LangChain — Models I/O

```python
from langchain.output_parsers.json import SimpleJsonOutputParser

# Create a JSON prompt
json_prompt = PromptTemplate.from_template(
    "Return a JSON object with `birthdate` and `birthplace` key that answers the following question: {question}"
)

# Initialize the JSON parser
json_parser = SimpleJsonOutputParser()

# Create a chain with the prompt, model, and parser
json_chain = json_prompt | model | json_parser

# Stream through the results
result_list = list(json_chain.stream({"question": "When and where was Elon Musk born?"}))

# The result is a list of JSON-like dictionaries
print(result_list)
```

```
[{'birthdate': 'June 28, 1971', 'birthplace': 'Pretoria, South Africa'}]
```

- Langchain's **SimpleJsonOutputParser** is used when you want to parse JSON-like outputs.

Building Blocks of LangChain — Retrieval



- ❑ Retrieve relevant external data and pass it to the language model
- ❑ Grounding the models on relevant and accurate information
- ❑ Documents are converted into their embeddings and stored in vector databases.

Building Blocks of LangChain — Retrieval

```
from langchain.document_loaders import PyPDFLoader

loader = PyPDFLoader("bcg-2022-annual-sustainability-report-apr-2023.pdf")
pdfpages = loader.load_and_split()
```

- ❑ **Document Loaders**

```
from langchain.text_splitter import RecursiveCharacterTextSplitter

state_of_the_union = "Your long text here..."

text_splitter = RecursiveCharacterTextSplitter(
    chunk_size=100,
    chunk_overlap=20,
    length_function=len,
    add_start_index=True,
)

texts = text_splitter.create_documents([state_of_the_union])
print(texts[0])
print(texts[1])
```

- ❑ **Document Transformers**
- ❑ *RecursiveCharacterTextSplitter*, a versatile text splitter that uses a character list for splitting. It allows parameters like chunk size, overlap, and starting index.

Building Blocks of LangChain — Retrieval

```python
from langchain.embeddings import OpenAIEmbeddings

# Initialize the model
embeddings_model = OpenAIEmbeddings()

# Embed a list of texts
embeddings = embeddings_model.embed_documents(
    ["Hi there!", "Oh, hello!", "What's your name?", "My friends call me World", '
)
print("Number of documents embedded:", len(embeddings))
print("Dimension of each embedding:", len(embeddings[0]))
```

- ***Text Embedding Models***

  embed_documents method is used to embed multiple texts, providing a list of vector representations.

```python
from langchain.embeddings import OpenAIEmbeddings

# Initialize the model
embeddings_model = OpenAIEmbeddings()

# Embed a single query
embedded_query = embeddings_model.embed_query("What was the name mentioned in the
print("First five dimensions of the embedded query:", embedded_query[:5])
```

- ***Text Embedding Models***

  embed_query is useful for comparing a query to a set of document embeddings.

Building Blocks of LangChain — Retrieval

```
from langchain.vectorstores import Chroma

db = Chroma.from_texts(embedded_texts)
similar_texts = db.similarity_search("search query")
```

```
from langchain.embeddings import OpenAIEmbeddings
from langchain.vectorstores import FAISS

pdfstore = FAISS.from_documents(pdfpages,
            embedding=OpenAIEmbeddings())

airtablestore = FAISS.from_documents(airtabledocs,
            embedding=OpenAIEmbeddings())
```

- ***Vector Stores***

  After embedding texts, we can store them in a vector store like *Chroma* and perform similarity searches

- ***Indexing***

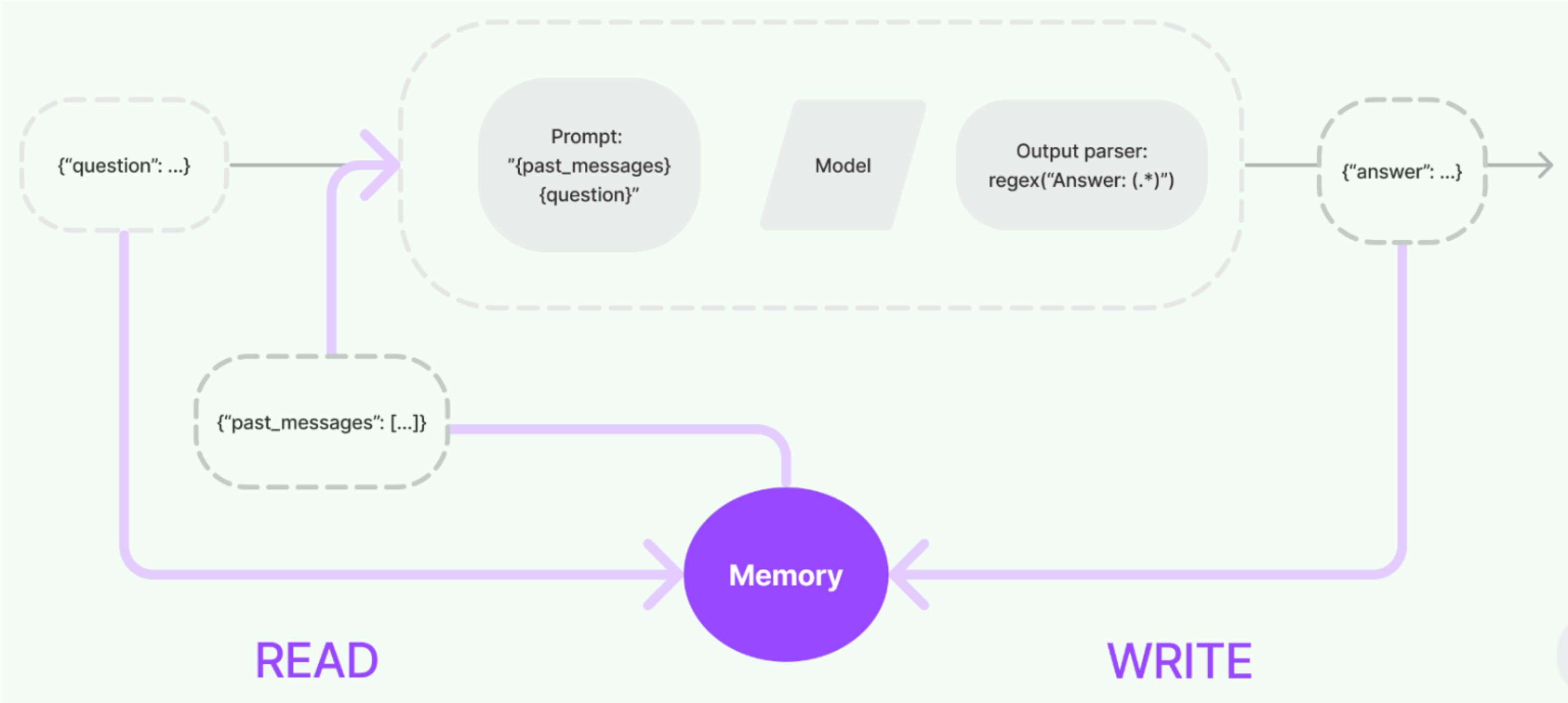
  use the *FAISS* vector store to create indexes for our documents.

Building Blocks of LangChain — Retrieval

```
from langchain.retrievers import BM25Retriever, EnsembleRetriever
from langchain.vectorstores import FAISS

bm25_retriever = BM25Retriever.from_texts(doc_list).set_k(2)
faiss_vectorstore = FAISS.from_texts(doc_list, OpenAIEmbeddings())
faiss_retriever = faiss_vectorstore.as_retriever(search_kwargs={"k": 2})

ensemble_retriever = EnsembleRetriever(
    retrievers=[bm25_retriever, faiss_retriever], weights=[0.5, 0.5]
)

docs = ensemble_retriever.get_relevant_documents("apples")
print(docs[0].page_content)
```

- ***Retrievers*** The *EnsembleRetriever* combines different retrieval algorithms to achieve better performance. An example of combining BM25 and FAISS Retrievers is shown in the above

Building Blocks of LangChain — Memory



- ❑ Refer to information introduced earlier in the conversation.

Building Blocks of LangChain — Memory

```
from langchain.llms import OpenAI
from langchain.prompts import PromptTemplate
from langchain.chains import LLMChain
from langchain.memory import ConversationBufferMemory

llm = OpenAI(temperature=0)
template = "Your conversation template here..."
prompt = PromptTemplate.from_template(template)
memory = ConversationBufferMemory(memory_key="chat_history")
conversation = LLMChain(llm=llm, prompt=prompt, memory=memory)

response = conversation({"question": "What's the weather like?"})
```

- LangChain's memory system integrates with its chains to provide a coherent and contextually aware conversational experience.

Application (ALL in ONE) —VectorStoreRetrieverMemory

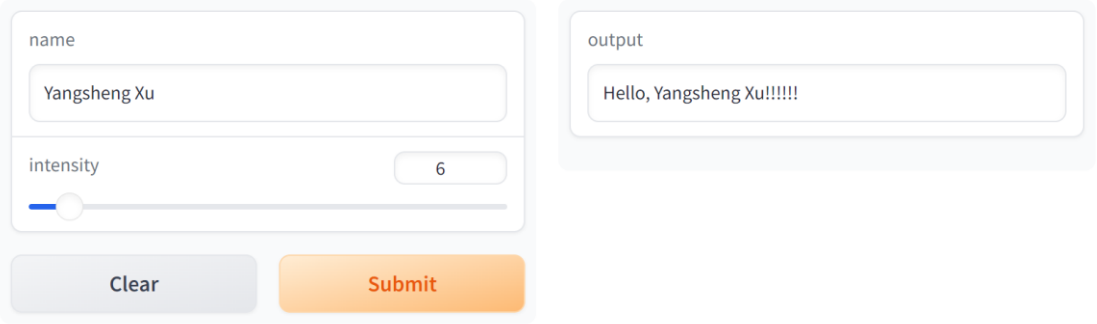
```
from datetime import datetime
from langchain.embeddings.openai import OpenAIEmbeddings
from langchain.llms import OpenAI
from langchain.memory import VectorStoreRetrieverMemory
from langchain.chains import ConversationChain
from langchain.prompts import PromptTemplate

# Initialize your vector store (specifics depend on the chosen vector store)
import faiss
from langchain.docstore import InMemoryDocstore
from langchain.vectorstores import FAISS

embedding_size = 1536  # Dimensions of the OpenAIEmbeddings
index = faiss.IndexFlatL2(embedding_size)
embedding_fn = OpenAIEmbeddings().embed_query
vectorstore = FAISS(embedding_fn, index, InMemoryDocstore({}), {})

# Create your VectorStoreRetrieverMemory
retriever = vectorstore.as_retriever(search_kwargs=dict(k=1))
memory = VectorStoreRetrieverMemory(retriever=retriever)

# Save context and relevant information to the memory
memory.save_context({"input": "My favorite food is pizza"}, {"output": "that's good to k
memory.save_context({"input": "My favorite sport is soccer"}, {"output": "..."})
memory.save_context({"input": "I don't like the Celtics"}, {"output": "ok"})

# Retrieve relevant information from memory based on a query
print(memory.load_memory_variables({"prompt": "what sport should i watch?"})["history"])
```

- *VectorStoreRetrieverMemor*
  - stores memories in a vector store
  - queries the top-K most "salient" documents every time it is called.

UI（Gradio）

```python
import gradio as gr

def greet(name, intensity):
    return "Hello, " + name + "!" * int(intensity)

demo = gr.Interface(
    fn=greet,
    inputs=["text", "slider"],
    outputs=["text"],
)

demo.launch()
```

```python
import gradio as gr

def greet(name, intensity):
    return "Hello, " + name + "!" * int(intensity)

demo = gr.Interface(
    fn=greet,
    inputs=["text", "slider"],
    outputs=["text"],
)

demo.launch()
```



https://www.gradio.app/guides/quickstart

Discussion