EasyInstruct: An Easy-to-use Instruction Processing Framework for Large Language Models

Yixin Ou♠, Ningyu Zhang♠*, Honghao Gui♠, Ziwen Xu♠, Shuofei Qiao♠, Yida Xue♠, Runnan Fang♠, Kangwei Liu♠, Lei Li♠, Zhen Bi♠, Guozhou Zheng♠, Huajun Chen♠*

♠ Zhejiang University

{ouyixin,zhangningyu,huajunsir}@zju.edu.cn

https://zjunlp.github.io/project/EasyInstruct

Abstract

In recent years, instruction tuning has gained increasing attention and emerged as a crucial technique to enhance the capabilities of Large Language Models (LLMs). To construct high-quality instruction datasets, many instruction processing approaches have been proposed, aiming to achieve a delicate balance between data quantity and data quality. Nevertheless, due to inconsistencies that persist among various instruction processing methods, there is no standard open-source instruction processing implementation framework available for the community, which hinders practitioners from further developing and advancing. To facilitate instruction processing research and development, we present EasyInstruct[1], an easy-to-use instruction processing framework for LLMs, which modularizes instruction generation, selection, and prompting, while also considering their combination and interaction. EasyInstruct is publicly released and actively maintained at https://github.com/zjunlp/EasyInstruct, along with an online demo app[2] and a demo video[3] for quick-start, calling for broader research centered on instruction data and synthetic data.

1 Introduction

Large Language Models (LLMs) have brought about a revolutionary transformation in the field of Natural Language Processing (NLP), leading to substantial improvement in performance across various tasks (Brown et al., 2020; OpenAI, 2023; Anil et al., 2023; Touvron et al., 2023b; Zhao et al., 2023; Chen et al., 2022; Qiao et al., 2023; Chen, 2023). To optimize the performance of LLMs in specific tasks or domains, it is crucial to adapt their outputs to specific contexts or instructions. Recent studies (Wei et al., 2022; Ouyang et al., 2022; Chung et al., 2022) have proposed instruction tuning methods for fine-tuning LLMs, which is a prominent research area aimed at optimizing the LLMs' behavior by providing explicit instructions during training, enabling better control and alignment with user preferences and desired outputs. Instruction dataset construction, which is also referred to as data engineering or management, poses a significant challenge in the process of instruction tuning (Zhao et al., 2023; Zhang et al., 2023; Wang et al., 2023c,d).

Substantial efforts have been dedicated to the task of construction instruction data through human annotations (Wang et al., 2022; Köpf et al., 2023), requiring a significant allocation of resources. Against this backdrop, LLMs are utilized to synthesize large-scale instruction data automatically (Wang et al., 2023b; Xu et al., 2023; Li et al., 2023b). These methods could scale up the size of instruction-following data, but they still inevitably suffer limited diversity and complexity, resulting in an unbalanced distribution and poor quality of instruction data. Recent studies (Zhou et al., 2023; Chen et al., 2023a; Xu et al., 2023) have unveiled a seminal revelation, indicating that even a small quantity of high-quality instruction data has the potential to yield robust performance. In general, instruction processing is an important process requiring careful attention to detail and rigorous quality assurance procedures to construct a high-quality instruction dataset for LLMs.

Unfortunately, the availability of open-source tools for instruction processing remains limited, especially in comparison to many open-source projects on models and training infrastructures (Touvron et al., 2023a,b; Taori et al., 2023; Scao et al., 2022; Chiang et al., 2023; Zeng

* Corresponding Author.

[1]This is a subproject of KnowLM (https://github.com/zjunlp/KnowLM), which facilitates knowledgeable LLM Framework with EasyInstruct, EasyEdit (Wang et al., 2023a; Yao et al., 2023; Zhang et al., 2024), EasyDetect etc.

[2]https://huggingface.co/spaces/zjunlp/EasyInstruct

[3]https://youtu.be/rfQOWYfziFo

Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 3: System Demonstrations), pages 94–106

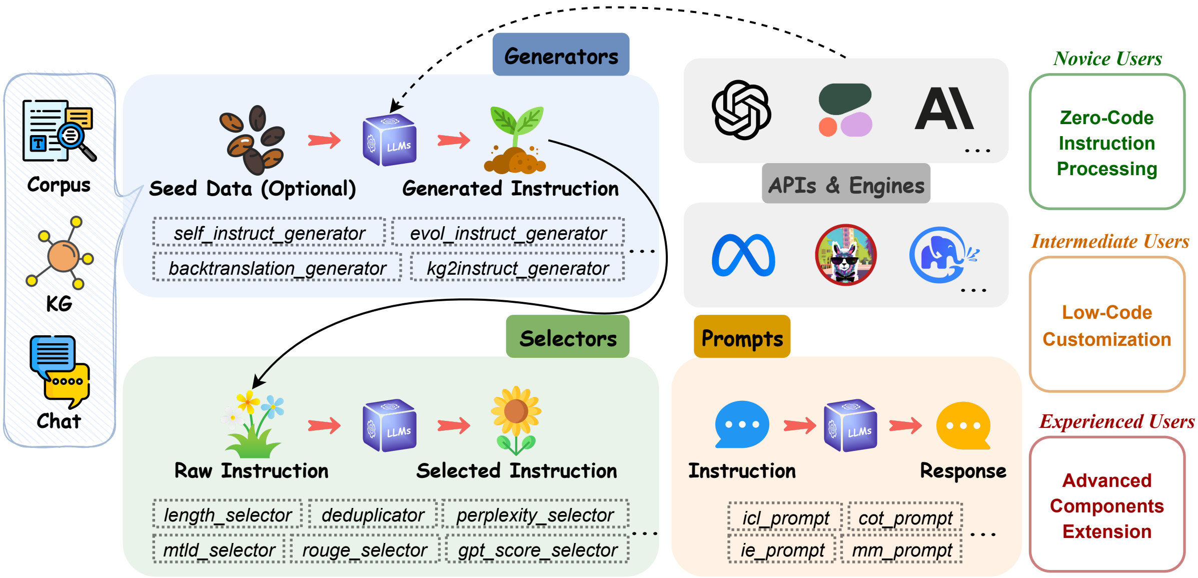


Figure 1: Overview of **EasyInstruct**. The `APIs & Engines` module standardizes the instruction execution process, enabling the execution of instruction prompts on the LLM API services or locally deployed LLMs. The `Generators` module streamlines the instruction generation process, enabling automated generation of instruction data based on **chat data, corpus, or Knowledge Graphs**. The `Selectors` module standardizes the instruction selection process, which enables the extraction of high-quality instruction datasets from raw, unprocessed instruction data. The `Prompts` module standardizes the instruction prompting process.

et al., 2023). Existing projects are often highly-customized to their own needs, lacking a systematized and modular processing ability to address diverse processing pipelines for LLMs. For instance, the Alpaca (Taori et al., 2023) dataset targets the augmentation of diversity for LLaMA tuning, whereas AlpaGasus (Chen et al., 2023a) focuses on filtering out low-quality instances from Alpaca. Thorough development of instruction processing systems for the ever-evolving and emerging requirements of LLM remains unexplored, particularly in light of the quick expansion of inventive LLM applications spanning various fields.

To address this issue, we develop EasyInstruct as depicted in Figure 1, an easy-to-use instruction processing framework for LLMs. Given some existing **chat data, corpus, or Knowledge Graphs**, EasyInstruct can handle instruction generation, selection, and prompting processes, while also considering their combination and interaction. These consistencies facilitate further development and comparisons of various methods, thus promoting the advancement of better instruction processing work. We further conduct experiments with EasyInstruct to validate its effectiveness in instruction processing. Currently, EasyInstuct is open-sourced on GitHub and has already received over 300 stars. We are committed to the long-term maintenance of EasyInstruct, providing continuous support for new features to ensure its effectiveness as a framework for instruction processing and synthetic data generation (Bauer et al., 2024).

2 Background

LLMs typically undergo two stages of training: pre-training and fine-tuning (Zhao et al., 2023). Despite the fact that large-scale pretraining is the key of the model's proficiency in generating natural language responses, these pre-trained models can still struggle with comprehending human instructions accurately. To bridge the gap between the training objectives and human objectives, instruction tuning is introduced as a potent strategy to amplify the controllability and capabilities and of LLMs in interpreting and responding to instructions (Wei et al., 2022; Ouyang et al., 2022; Chung et al., 2022; Wang et al., 2023b; Zhang et al., 2023; Lou et al., 2023). Concretely, instruction tuning involves the method of refining pre-trained LLMs through supervised learning, utilizing examples structured as (INSTRUCTION, INPUT, OUTPUT). In this format, INSTRUCTION represents the human-given directive that outlines the task, INPUT optionally offers additional context, and OUTPUT signifies the expected outcome in alignment with the INSTRUCTION and any given INPUT.

Despite the effectiveness of instruction tuning, constructing high-quality large-scale instructions which effectively encompass the target behaviors remains a non-trivial challenge in this realm. Existing instruction datasets are often limited in terms of diversity, quantity, and creativity, which underscores the significance of instruction processing. One typical method for constructing instruction datasets is data integration. In this method, instructional datasets are constructed by merging existing annotated datasets with descriptions of tasks in natural language (Longpre et al., 2023; Sanh et al., 2022; Anand et al., 2023). Another prevalent method for constructing instruction datasets is automated generation. To alleviate the need for extensive human annotation or manual data gathering, automated methods have been proposed to generate large volumes of instructional data through the use of LLMs. Instructions can be sourced from chat data (Chiang et al., 2023) or expanded on a small set of seed instructions using LLMs (Wang et al., 2023b; Xu et al., 2023; Li et al., 2023b). Subsequently, the collected instructions are fed into LLMs to generate corresponding inputs and outputs. In EasyInstruct, our primary focus lies on automated approaches for instruction generation due to their high efficiency and scalability.

Another promising research direction of instruction processing is the selection of high-quality instruction. Recently, numerous studies (Zhou et al., 2023; Chen et al., 2023a; Xu et al., 2023; Liu et al., 2023) have investigated the issue of the scale of the instruction dataset for fine-tuning and have indicated that merely increasing the number of instructions may not necessarily result in enhancements. Instead, a modest volume of high-quality instruction data can influence the fine-tuning of LLMs, yielding solid performance. Thus, optimizing the instruction dataset and enhancing its quality play a critical role in fine-tuning LLMs effectively.

From a practical implementation point of view, instruction processing is actually complex and requires meticulous consideration. In this paper, we present **EasyInstruct**, an easy-to-use framework to effectively and efficiently implement instruction processing approaches including instruction generation, selection, and prompting. Through this framework, EasyInstruct can help users to quickly comprehend and apply the existing instruction processing methods implemented in the package.

3 Design and Implementation

As illustrated in Figure 1, EasyInstruct provides a complete instruction processing procedure built on PyTorch and Huggingface. In this section, we first introduce the design principles, and then detail the implementation of the major modules.

3.1 Design Principles

The framework is designed to cater to users with varying levels of expertise, providing a user-friendly experience ranging from code-free execution to low-code customization and advanced components extension options:

Zero-Code Instruction Processing. Novice users, who do not require coding knowledge, can leverage pre-defined configuration files and shell scripts to accomplish code-free instruction processing. By running these scripts, they can complete instruction processing tasks without the need for coding skills. Example configuration files and shell scripts are shown in Appendix A.2.1.

Low-Code Customization. Intermediate users have the option to customize various process inputs and outputs using a low-code approach. This allows them to have more control over the different stages within the framework. A running example is shown in Figure 2.

Advanced Components Extension. Experienced users can easily extend our components based on their specific scenarios and requirements. To customize their classes, users can inherit the base classes of modules and override the necessary methods as per their requirements. This flexibility enables them to implement their functional components, tailored to their unique needs.

3.2 APIs & Engines

The `APIs` modules integrate with mainstream LLMs, including API services provided by companies such as OpenAI[4], Anthropic[5], and Cohere[6]. This integration facilitates the seamless invocation of various relevant steps within the framework. We list a range of API service providers and their corresponding LLM products that are currently available in EasyInstruct in Appendix A.5. The `Engines` module standardizes the instruction execution process, which enables the execution of

[4] `https://platform.openai.com/docs`
[5] `https://docs.anthropic.com/claude/docs`
[6] `https://docs.cohere.com/docs`

instruction prompts on several open-source LLMs such as LLaMA (Touvron et al., 2023a,b) and ChatGLM (Du et al., 2022; Zeng et al., 2023).

3.3 Generators

The `Generators` module streamlines the process of instruction generation, enabling automated generation of instruction data based on seed data, where seed data can be sourced from either chat data, corpus, or Knowledge Graphs. As listed in Table 1, the instruction generation methods implemented in `Generators` are categorized into three groups, based on their respective seed data sources.

Chat Data. Early work (Wang et al., 2023b) randomly samples a few instructions from a human-annotated seed tasks pool as demonstrations and then, prompts an LLM to generate more instructions and corresponding input-output pairs. Due to its adaptability, *Self-Instruct* remains the prevailing preference among automated instruction generation methods. Similarly, starting with an initial set of instructions, *Evol-Instruct* (Xu et al., 2023) incrementally upgrades them into more complex instructions by prompting an LLM with specific prompts. In contrast to the *Self-Instruct* generation approach, *Evol-Instruct* allows for the adjustment of the difficulty and intricacy of the instructions it produces.

Corpus. Given an unannotated corpus, *Instruction Backtranslation* (Li et al., 2023b) creates an instruction following training instance by predicting an instruction that would be correctly answered by a paragraph in the document or corpus. Considering the mixed quality of human-written web text and the presence of noise in generated content, only the highest quality instances are reserved.

Knowledge Graphs. Incorporating existing knowledge graphs, *KG2Instruct* (Gui et al., 2023) generates Information Extraction (IE) instruction datasets. To enhance the generalizability of instructions, a random sampling approach is utilized based on human-crafted instruction templates.

EasyInstruct has implemented the existing methods above to facilitate future research and systematic comparison of automated generation of instruction data. Furthermore, the flexibility of the `Generators` module allows practitioners to select the appropriate generator and make further modification that best suits their specific needs. A running example of using a `Generator` class in EasyInstruct is shown in Figure 2.

```python
from easyinstruct import SelfInstructGenerator
from easyinstruct import GPTScoreSelector
from easyinstruct.utils.api import set_openai_key

# Step1: Set your own API-KEY
set_openai_key("YOUR-KEY")

# Step2: Declare a generator class
generator = SelfInstructGenerator(
    data_format = "alpaca",
    seed_tasks_path = "seed_tasks.jsonl",
    generated_instances_path = "generation.jsonl",
    num_instructions_to_generate=100,
    engine = "gpt-3.5-turbo",
)

# Step3: Generate self-instruct data
generator.generate()

# Step4: Declare a selector class
selector = GPTScoreSelector(
    source_file_path = "generation.jsonl",
    engine = "gpt-3.5-turbo",
    threshold = 4,
)

# Step5: Process raw data
selector.process()
```

Figure 2: A running example of instruction generation and selection in **EasyInstruct**.

3.4 Selectors

The `Selectors` module is designed to streamline the process of filtering instructions, enabling the curation of instruction datasets from raw instruction data. This raw data might originate from publicly accessible instruction datasets or be synthesised in advence by the `Generators` module. Table 1 provides a comprehensive overview of various metrics for instruction quality evaluation. We divide the evaluation metrics into four categories based on the principle of their implementation: statistics-based, n-gram-based, structure-based and LM-based. All `Selector` classes derive from a common base class, `BaseSelector`. It includes fundamental attributes and abstract methods such as loading, processing, and dumping of data. In EasyInstruct, multiple `Selectors` can be grouped for convenient usage, which allows users to achieve more concise and readable code. A running example of using a `Selector` class is shown in Figure 2.

3.5 Prompts

The `Prompts` module standardizes the instruction prompting step, in which user requests are constructed as instruction prompts and sent to specific LLMs to obtain responses. Utilizing the `Prompts` module with a series of well-designed and refined prompts enhances the ability of `Generators` and `Selectors` to effectively fulfill their respective functions. Similar to `Selectors`, all

Modules	Methods	Seed	Description
Generators	Self-Instruct	Chat	The method that randomly samples a few instructions as demonstrations and generates more instructions and input-output pairs using LLM (Wang et al., 2023b).
	Evol-Instruct	Chat	The method that incrementally upgrades an initial set of instructions into more complex instructions by prompting an LLM with specific prompts (Xu et al., 2023).
	Backtranslation	Corpus	The method that creates a training instance by predicting an instruction that would be correctly answered by a paragraph in the corpus (Li et al., 2023b).
	KG2Instruct	KG	The method that generates Information Extraction (IE) instruction datasets incorporating existing Knowledge Graphs (Gui et al., 2023).
Modules	**Metrics**	**Type**	**Description**
Selectors	Deduplication	Statistics-based	Repetitive input and output of instances.
	Length	Statistics-based	The bounded length of every pair of instruction and output.
	MTLD	Statistics-based	A metric for assessing the lexical diversity in text, defined as the average length of word sequences that sustain a minimum threshold TTR score (McCarthy and Jarvis, 2010).
	ROUGE	N-gram-based	Recall-oriented understudy for gisting evaluation (Lin, 2004).
	CIRS	Structure-based	The score using the abstract syntax tree to encode structural and logical attributes, to evaluate the correlation between code and reasoning abilities (Bi et al., 2023).
	Perplexity	LM-based	The exponentiated average negative log-likelihood of text.
	GPT Score	LM-based	The score that ChatGPT/GPT4 assigns to assess how effectively the AI Assistant's response aligns with the user's instructions.

Table 1: Components of `Generators` and `Selectors` modules of EasyInstruct. The instruction generation methods implemented in `Generators` are categorized into three groups, based on their respective seed data sources: chat data, corpus, and knowledge graphs. The evaluation metrics in `Selecors` are divided into four categories, based on the principle of their implementation: statistics-based, n-gram-based, structure-based, and LM-based.

`Prompts` classes inherit from a common base class, `BasePrompt`, which includes necessary attributes and abstract methods. In the mentioned base class, there are functionalities provided for building prompts, requesting generation results from LLMs, and parsing the responses received from LLMs. The base class also provides mechanisms to handle error conditions and exceptions that may occur during the whole process. Users can inherit from the base class and customize or extend its functionality based on their specific requirements. We also equip EasyInstruct with various prompting techniques and application adaptions (e.g. Chain-of-Thought, Information Extraction, Multimodal, etc.) by providing a consistent and standardized interface, enabling efficient instruction prompting for LLMs.

4 Evaluation

In terms of evaluation, we will introduce the experiment setups and illustrate the empirical results of multiple modules implemented in EasyInstruct to demonstrate its capability.

4.1 Experiment Setups

Instruction Datasets. We adopt the popular *Self-Instruct* (Wang et al., 2023b) and *Evol-Instruct* (Xu et al., 2023) methods implemented in EasyInstruct to synthesize instruction datasets, containing instructions paired with instance inputs and outputs separately. We mainly consider four instruction datasets as follows: (a) ***self_instruct_5k*** is constructed by employing the *Self-Instruct* method to distill instruction data from text-davinci-003; (b) ***alpaca_data_5k*** is randomly sampled from the Alpaca dataset (Taori et al., 2023); (c) ***evol_instruct_5k*** is constructed by employing the *Evol-Instruct* method; (d) ***easyinstruct_5k*** is collected by integrating the three instruction datasets above and applying multiple `Selectors` in EasyInstruct to extract high-quality instruction datasets. We detail the hyper-parameters for selection in a configuration file, as shown in Appendix A.2.1.

Implementation Setups. To conduct the experiments on the effect of instruction datasets, we adopt a LLaMA2 (7B) model (Touvron et al., 2023b), a popular open-sourced LLM backbone widely used for instruction-tuning. We fine-tune the LLMs with LoRA (Hu et al., 2022) in the format following Alpaca (Taori et al., 2023) with KnowLM[7]. All hyper-parameters for training can be found in Appendix A.3. The evaluation is conducted by comparing the

[7] https://github.com/zjunlp/KnowLM

[Instruction] Describe the common theme of the following three animals.

[Output] The common theme among the penguin, polar bear and walrus is their ability to survive in cold climates. All three animals have adapted to thrive in cold Arctic regions, with a heavy layer of blubber or thick fur to keep them warm and plenty of blubber to provide sustenance.

[Scores] {"mtld_score": 22.68, "avg_rouge_score": 0.16, "ppl_score": 22.57, "gpt_score": 4}

[Instruction] Pretend that you are the subject of the following poem. Use the first person voice to write a response to the poem.

[Input] The gentle swell of the sea,\n Cradling the shore in its serenity.

[Output] I can hear the sound of the waves crashing against the shore. The sea's serenity is like a soothing cradle that brings me peace and tranquility.

[Scores] {"mtld_score": 27.10, "avg_rouge_score": 0.16, "ppl_score": 26.02, "gpt_score": 4}

[Instruction] Create a function to calculate the total cost of a book order, given the unit price, quantity, shipping cost, and tax rate.

[Input] Unit Price: 10.99\n Quantity: 3\n Shipping Cost: 5\n Tax Rate: 0.03

[Output]
```
def calculateCost(unitPrice, quantity, shippingCost, taxRate):
    totalPrice = (unitPrice * quantity) + shippingCost
    totalTax = totalPrice * taxRate
    totalCost = totalPrice + totalTax
    return totalCost
```

[Scores] {"mtld_score": 67.76, "avg_rouge_score": 0.17, "ppl_score": 18.95, "gpt_score": 5}

Table 2: Examples of the selected instructions in ***easyinstruct_5k***, and their corresponding evaluation scores given by the `Selectors` module.

generated results from different fine-tuned models based on the AlpacaFarm (Dubois et al., 2023) evaluation set. Following AlpacaFarm, for each comparison, we employ ChatGPT as the evaluator to automatically compare two outputs from different models and label which one they prefer, reporting the win rate as the evaluation metric. For both instruction tuning and evaluation, we adopt the same prompt templates used by Alpaca-LoRA[8], as shown in Appendix A.4.

4.2 Experiment Results

Main Results. We compare the generated outputs from models fine-tuned separately on the four instruction datasets with the outputs from the base version of the LLaMA2 (7B) model on the AlpacaFarm evaluation set. As depicted in Figure 3, there are improvements in the win rate metric for all the settings. Moreover, the model performs optimally under the ***easyinstruct_5k*** setting, indicating the importance of a rich instruction selection strategy.

Instruction Diversity. To study the diversity of the instruction datasets considered in our experiments, we identify the verb-noun structure in the generated instructions and plot the top 20 most prevalent root verbs and their top 4 direct nouns

[8]`https://github.com/tloen/alpaca-lora`

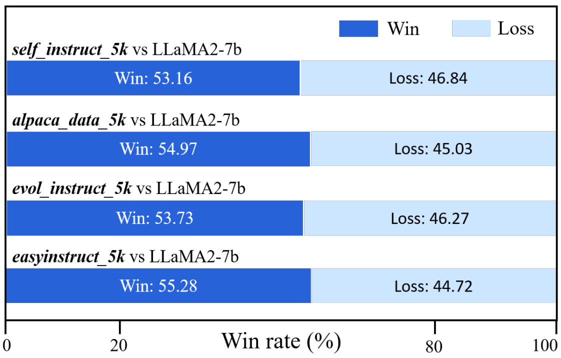
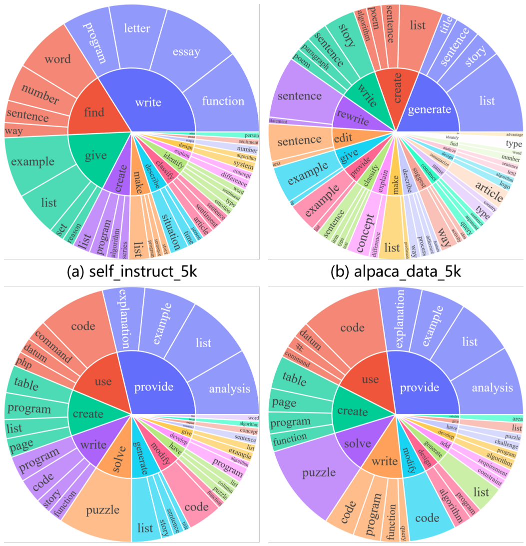


Figure 3: Results of models fine-tuned on four distinct instruction datasets against those from the base LLaMA2 (7B) model, using the AlpacaFarm evaluation set for assessment.

in Figure 4, following the approach of Wang et al. (2023b). Overall, we see a wide range of intents and textual formats within these instructions.

Case Study. To conduct a qualitative evaluation of EasyInstruct, we sample several instruction examples selected by the `Selectors` module in ***easyinstruct_5k*** for the case study. We also attach the corresponding evaluation scores for each of these instruction examples, as shown in Table 2. We observe that the selected instructions often possess fluent language and meticulous logic.



(a) self_instruct_5k (b) alpaca_data_5k (c) evol_instruct_5k (d) easyinstruct_5k

Figure 4: (Inner circle refers to the top 20 most prevalent root verbs and outer circle indicates their top 4 direct nouns in the generated instruction datasets considered in the experiments.

5 Conclusion and Future Work

We present **EasyInstruct**, an easy-to-use instruction processing framework for LLMs. EasyInstruct can combine chat data, corpus, KGs and LLMs as an automated instruction generation tool, reducing the cost of manual data annotation. Additionally, EasyInstruct integrates a diverse set of instruction selection tools to optimize the diversity and distribution of instruction data, thereby improving the quality of fine-tuning data. EasyInstruct is designed to be easy to extend, and we will continue to update new features (e.g., knowledgeable synthetic data generation) to keep pace with the latest research. We expect EasyInstruct to be a helpful framework for researchers and practitioners to facilitate their work of instruction tuning on LLMs.

Limitations

In this paper, we are committed to unifying all phases of instruction data processing including instruction generation, selection, and prompting. Despite our efforts, this paper may still have some remaining limitations.

The Scope of Instruction Selection Methods. We implement various instruction selection methods within the `Selectors` module. Based on the evaluation metrics utilized and the model base employed, the implemented instruction data selection methods can be divided into three categories: methods based on a system of indicators, methods utilizing powerful LLMs like ChatGPT, and methods employing small models (Wang et al., 2024). However, another line of work (Li et al., 2023a,c,b; Wu et al., 2023; Chen et al., 2023b; Kung et al., 2023) employs trainable LLMs like LLaMA for computation formulas in instruction selection processes, which are not integrated into the `Selectors` module. Although our design choice is to decouple instruction processing and model training into two separate phases, we regard it as a limitation that may be addressed by future work.

Statistics for evaluating efficiency. In our evaluation, we fine-tune a LLaMA2 (7B) model utilizing multiple modules implemented in EasyInstruct. Compared to models fine-tuned on other instruction datasets constructed without EasyInstruct, our model achieves optimal results, demonstrating EasyInstruct's capability. Although we also qualitatively demonstrate the ease of writing code for instruction processing with multiple code samples and configuration files using EasyInstruct, a limitation is the lack of appropriate statistics for quantitatively evaluating efficiency.

Acknowledgments

We would like to express gratitude to the anonymous reviewers for their kind comments. This work was supported by the National Natural Science Foundation of China (No. 62206246, No. NSFCU23B2055, No. NSFCU19B2027), the Fundamental Research Funds for the Central Universities (226-2023-00138), Zhejiang Provincial Natural Science Foundation of China (No. LGG22F030011), Yongjiang Talent Introduction Programme (2021A-156-G), CCF-Baidu Open Fund, Information Technology Center and State Key Lab of CAD&CG, Zhejiang University.

References

Yuvanesh Anand, Zach Nussbaum, Brandon Duderstadt, Benjamin Schmidt, and Andriy Mulyar. 2023. Gpt4all: Training an assistant-style chatbot with large scale data distillation from gpt-3.5-turbo. https://github.com/nomic-ai/gpt4all.

Rohan Anil, Andrew M. Dai, Orhan Firat, Melvin Johnson, Dmitry Lepikhin, Alexandre Passos, Siamak Shakeri, Emanuel Taropa, Paige Bailey, Zhifeng Chen, Eric Chu, Jonathan H. Clark, Laurent El Shafey, Yanping Huang, Kathy Meier-Hellstern, Gaurav Mishra, Erica Moreira, Mark Omernick, Kevin

Robinson, Sebastian Ruder, Yi Tay, Kefan Xiao, Yuanzhong Xu, Yujing Zhang, Gustavo Hernández Ábrego, Junwhan Ahn, Jacob Austin, Paul Barham, Jan A. Botha, James Bradbury, Siddhartha Brahma, Kevin Brooks, Michele Catasta, Yong Cheng, Colin Cherry, Christopher A. Choquette-Choo, Aakanksha Chowdhery, Clément Crepy, Shachi Dave, Mostafa Dehghani, Sunipa Dev, Jacob Devlin, Mark Díaz, Nan Du, Ethan Dyer, Vladimir Feinberg, Fangxiaoyu Feng, Vlad Fienber, Markus Freitag, Xavier Garcia, Sebastian Gehrmann, Lucas Gonzalez, and et al. 2023. Palm 2 technical report. *CoRR*, abs/2305.10403.

André Bauer, Simon Trapp, Michael Stenger, Robert Leppich, Samuel Kounev, Mark Leznik, Kyle Chard, and Ian T. Foster. 2024. Comprehensive exploration of synthetic data generation: A survey. *CoRR*, abs/2401.02524.

Zhen Bi, Ningyu Zhang, Yinuo Jiang, Shumin Deng, Guozhou Zheng, and Huajun Chen. 2023. When do program-of-thoughts work for reasoning? *CoRR*, abs/2308.15452.

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language models are few-shot learners. In *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*.

Huajun Chen. 2023. Large knowledge model: Perspectives and challenges. *CoRR*, abs/2312.02706.

Lichang Chen, Shiyang Li, Jun Yan, Hai Wang, Kalpa Gunaratna, Vikas Yadav, Zheng Tang, Vijay Srinivasan, Tianyi Zhou, Heng Huang, and Hongxia Jin. 2023a. Alpagasus: Training A better alpaca with fewer data. *CoRR*, abs/2307.08701.

Xiang Chen, Ningyu Zhang, Xin Xie, Shumin Deng, Yunzhi Yao, Chuanqi Tan, Fei Huang, Luo Si, and Huajun Chen. 2022. Knowprompt: Knowledge-aware prompt-tuning with synergistic optimization for relation extraction. In *WWW '22: The ACM Web Conference 2022, Virtual Event, Lyon, France, April 25 - 29, 2022*, pages 2778–2788. ACM.

Yongrui Chen, Haiyun Jiang, Xinting Huang, Shuming Shi, and Guilin Qi. 2023b. Tegit: Generating high-quality instruction-tuning data with text-grounded task design. *CoRR*, abs/2309.05447.

Wei-Lin Chiang, Zhuohan Li, Zi Lin, Ying Sheng, Zhanghao Wu, Hao Zhang, Lianmin Zheng, Siyuan Zhuang, Yonghao Zhuang, Joseph E. Gonzalez, Ion Stoica, and Eric P. Xing. 2023. Vicuna: An open-source chatbot impressing gpt-4 with 90%* chatgpt quality.

Hyung Won Chung, Le Hou, Shayne Longpre, Barret Zoph, Yi Tay, William Fedus, Eric Li, Xuezhi Wang, Mostafa Dehghani, Siddhartha Brahma, Albert Webson, Shixiang Shane Gu, Zhuyun Dai, Mirac Suzgun, Xinyun Chen, Aakanksha Chowdhery, Sharan Narang, Gaurav Mishra, Adams Yu, Vincent Y. Zhao, Yanping Huang, Andrew M. Dai, Hongkun Yu, Slav Petrov, Ed H. Chi, Jeff Dean, Jacob Devlin, Adam Roberts, Denny Zhou, Quoc V. Le, and Jason Wei. 2022. Scaling instruction-finetuned language models. *CoRR*, abs/2210.11416.

Zhengxiao Du, Yujie Qian, Xiao Liu, Ming Ding, Jiezhong Qiu, Zhilin Yang, and Jie Tang. 2022. Glm: General language model pretraining with autoregressive blank infilling. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 320–335.

Yann Dubois, Xuechen Li, Rohan Taori, Tianyi Zhang, Ishaan Gulrajani, Jimmy Ba, Carlos Guestrin, Percy Liang, and Tatsunori B. Hashimoto. 2023. Alpacafarm: A simulation framework for methods that learn from human feedback. *CoRR*, abs/2305.14387.

Honghao Gui, Jintian Zhang, Hongbin Ye, and Ningyu Zhang. 2023. Instructie: A chinese instruction-based information extraction dataset. *CoRR*, abs/2305.11527.

Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2022. Lora: Low-rank adaptation of large language models. In *The Tenth International Conference on Learning Representations, ICLR 2022, Virtual Event, April 25-29, 2022*. OpenReview.net.

Andreas Köpf, Yannic Kilcher, Dimitri von Rütte, Sotiris Anagnostidis, Zhi-Rui Tam, Keith Stevens, Abdullah Barhoum, Nguyen Minh Duc, Oliver Stanley, Richárd Nagyfi, Shahul ES, Sameer Suri, David Glushkov, Arnav Dantuluri, Andrew Maguire, Christoph Schuhmann, Huu Nguyen, and Alexander Mattick. 2023. Openassistant conversations - democratizing large language model alignment. *CoRR*, abs/2304.07327.

Po-Nien Kung, Fan Yin, Di Wu, Kai-Wei Chang, and Nanyun Peng. 2023. Active instruction tuning: Improving cross-task generalization by training on prompt sensitive tasks. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing, EMNLP 2023, Singapore, December 6-10, 2023*, pages 1813–1829. Association for Computational Linguistics.

Ming Li, Yong Zhang, Zhitao Li, Jiuhai Chen, Lichang Chen, Ning Cheng, Jianzong Wang, Tianyi Zhou, and Jing Xiao. 2023a. From quantity to quality: Boosting LLM performance with self-guided data selection for instruction tuning. *CoRR*, abs/2308.12032.

Xian Li, Ping Yu, Chunting Zhou, Timo Schick, Luke Zettlemoyer, Omer Levy, Jason Weston, and Mike Lewis. 2023b. Self-alignment with instruction backtranslation. *CoRR*, abs/2308.06259.

Yunshui Li, Binyuan Hui, Xiaobo Xia, Jiaxi Yang, Min Yang, Lei Zhang, Shuzheng Si, Junhao Liu, Tongliang Liu, Fei Huang, and Yongbin Li. 2023c. One shot learning as instruction data prospector for large language models. *CoRR*, abs/2312.10302.

Chin-Yew Lin. 2004. ROUGE: A package for automatic evaluation of summaries. In *Text Summarization Branches Out*, pages 74–81, Barcelona, Spain. Association for Computational Linguistics.

Wei Liu, Weihao Zeng, Keqing He, Yong Jiang, and Junxian He. 2023. What makes good data for alignment? A comprehensive study of automatic data selection in instruction tuning. *CoRR*, abs/2312.15685.

Shayne Longpre, Le Hou, Tu Vu, Albert Webson, Hyung Won Chung, Yi Tay, Denny Zhou, Quoc V. Le, Barret Zoph, Jason Wei, and Adam Roberts. 2023. The flan collection: Designing data and methods for effective instruction tuning. In *International Conference on Machine Learning, ICML 2023, 23-29 July 2023, Honolulu, Hawaii, USA*, volume 202 of *Proceedings of Machine Learning Research*, pages 22631–22648. PMLR.

Renze Lou, Kai Zhang, and Wenpeng Yin. 2023. Is prompt all you need? no. A comprehensive and broader view of instruction learning. *CoRR*, abs/2303.10475.

Philip M McCarthy and Scott Jarvis. 2010. Mtld, vocd-d, and hd-d: A validation study of sophisticated approaches to lexical diversity assessment. *Behavior research methods*, 42(2):381–392.

OpenAI. 2023. GPT-4 technical report. *CoRR*, abs/2303.08774.

Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll L. Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, John Schulman, Jacob Hilton, Fraser Kelton, Luke Miller, Maddie Simens, Amanda Askell, Peter Welinder, Paul F. Christiano, Jan Leike, and Ryan Lowe. 2022. Training language models to follow instructions with human feedback. In *NeurIPS*.

Shuofei Qiao, Yixin Ou, Ningyu Zhang, Xiang Chen, Yunzhi Yao, Shumin Deng, Chuanqi Tan, Fei Huang, and Huajun Chen. 2023. Reasoning with language model prompting: A survey. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2023, Toronto, Canada, July 9-14, 2023*, pages 5368–5393. Association for Computational Linguistics.

Victor Sanh, Albert Webson, Colin Raffel, Stephen H. Bach, Lintang Sutawika, Zaid Alyafeai, Antoine Chaffin, Arnaud Stiegler, Arun Raja, Manan Dey, M Saiful Bari, Canwen Xu, Urmish Thakker, Shanya Sharma Sharma, Eliza Szczechla, Taewoon Kim, Gunjan Chhablani, Nihal V. Nayak, Debajyoti Datta, Jonathan Chang, Mike Tian-Jian Jiang, Han Wang, Matteo Manica, Sheng Shen, Zheng Xin Yong, Harshit Pandey, Rachel Bawden, Thomas Wang, Trishala Neeraj, Jos Rozen, Abheesht Sharma, Andrea Santilli, Thibault Févry, Jason Alan Fries, Ryan Teehan, Teven Le Scao, Stella Biderman, Leo Gao, Thomas Wolf, and Alexander M. Rush. 2022. Multitask prompted training enables zero-shot task generalization. In *The Tenth International Conference on Learning Representations, ICLR 2022, Virtual Event, April 25-29, 2022*. OpenReview.net.

Teven Le Scao, Angela Fan, Christopher Akiki, Ellie Pavlick, Suzana Ilic, Daniel Hesslow, Roman Castagné, Alexandra Sasha Luccioni, François Yvon, Matthias Gallé, Jonathan Tow, Alexander M. Rush, Stella Biderman, Albert Webson, Pawan Sasanka Ammanamanchi, Thomas Wang, Benoît Sagot, Niklas Muennighoff, Albert Villanova del Moral, Olatunji Ruwase, Rachel Bawden, Stas Bekman, Angelina McMillan-Major, Iz Beltagy, Huu Nguyen, Lucile Saulnier, Samson Tan, Pedro Ortiz Suarez, Victor Sanh, Hugo Laurençon, Yacine Jernite, Julien Launay, Margaret Mitchell, Colin Raffel, Aaron Gokaslan, Adi Simhi, Aitor Soroa, Alham Fikri Aji, Amit Alfassy, Anna Rogers, Ariel Kreisberg Nitzav, Canwen Xu, Chenghao Mou, Chris Emezue, Christopher Klamm, Colin Leong, Daniel van Strien, David Ifeoluwa Adelani, and et al. 2022. BLOOM: A 176b-parameter open-access multilingual language model. *CoRR*, abs/2211.05100.

Rohan Taori, Ishaan Gulrajani, Tianyi Zhang, Yann Dubois, Xuechen Li, Carlos Guestrin, Percy Liang, and Tatsunori B. Hashimoto. 2023. Stanford alpaca: An instruction-following llama model. `https://github.com/tatsu-lab/stanford_alpaca`.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurélien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. 2023a. Llama: Open and efficient foundation language models. *CoRR*, abs/2302.13971.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, Dan Bikel, Lukas Blecher, Cristian Canton-Ferrer, Moya Chen, Guillem Cucurull, David Esiobu, Jude Fernandes, Jeremy Fu, Wenyin Fu, Brian Fuller, Cynthia Gao, Vedanuj Goswami, Naman Goyal, Anthony Hartshorn, Saghar Hosseini, Rui Hou, Hakan Inan, Marcin Kardas, Viktor Kerkez, Madian Khabsa, Isabel Kloumann, Artem Korenev, Punit Singh Koura, Marie-Anne Lachaux, Thibaut Lavril, Jenya Lee, Diana Liskovich, Yinghai Lu, Yuning Mao, Xavier Martinet, Todor Mihaylov, Pushkar Mishra, Igor Molybog, Yixin Nie, Andrew Poulton, Jeremy Reizenstein, Rashi Rungta, Kalyan Saladi, Alan Schelten, Ruan Silva, Eric Michael Smith, Ranjan Subramanian, Xiaoqing Ellen Tan, Binh Tang, Ross Taylor, Adina Williams, Jian Xiang Kuan, Puxin Xu, Zheng Yan, Iliyan Zarov, Yuchen Zhang, Angela Fan, Melanie Kambadur, Sharan Narang, Aurélien Rodriguez, Robert Stojnic, Sergey Edunov, and Thomas Scialom. 2023b. Llama 2: Open foundation and fine-tuned chat models. *CoRR*, abs/2307.09288.

Jiahao Wang, Bolin Zhang, Qianlong Du, Jiajun Zhang, and Dianhui Chu. 2024. A survey on data selection for LLM instruction tuning. *CoRR*, abs/2402.05123.

Peng Wang, Ningyu Zhang, Xin Xie, Yunzhi Yao, Bozhong Tian, Mengru Wang, Zekun Xi, Siyuan Cheng, Kangwei Liu, Guozhou Zheng, and Huajun Chen. 2023a. Easyedit: An easy-to-use knowledge editing framework for large language models. *CoRR*, abs/2308.07269.

Yizhong Wang, Yeganeh Kordi, Swaroop Mishra, Alisa Liu, Noah A. Smith, Daniel Khashabi, and Hannaneh Hajishirzi. 2023b. Self-instruct: Aligning language models with self-generated instructions. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2023, Toronto, Canada, July 9-14, 2023*, pages 13484–13508. Association for Computational Linguistics.

Yizhong Wang, Swaroop Mishra, Pegah Alipoormolabashi, Yeganeh Kordi, Amirreza Mirzaei, Atharva Naik, Arjun Ashok, Arut Selvan Dhanasekaran, Anjana Arunkumar, David Stap, Eshaan Pathak, Giannis Karamanolakis, Haizhi Gary Lai, Ishan Purohit, Ishani Mondal, Jacob Anderson, Kirby Kuznia, Krima Doshi, Kuntal Kumar Pal, Maitreya Patel, Mehrad Moradshahi, Mihir Parmar, Mirali Purohit, Neeraj Varshney, Phani Rohitha Kaza, Pulkit Verma, Ravsehaj Singh Puri, Rushang Karia, Savan Doshi, Shailaja Keyur Sampat, Siddhartha Mishra, Sujan Reddy A, Sumanta Patro, Tanay Dixit, and Xudong Shen. 2022. Super-naturalinstructions: Generalization via declarative instructions on 1600+ NLP tasks. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing, EMNLP 2022, Abu Dhabi, United Arab Emirates, December 7-11, 2022*, pages 5085–5109. Association for Computational Linguistics.

Yufei Wang, Wanjun Zhong, Liangyou Li, Fei Mi, Xingshan Zeng, Wenyong Huang, Lifeng Shang, Xin Jiang, and Qun Liu. 2023c. Aligning large language models with human: A survey. *CoRR*, abs/2307.12966.

Zige Wang, Wanjun Zhong, Yufei Wang, Qi Zhu, Fei Mi, Baojun Wang, Lifeng Shang, Xin Jiang, and Qun Liu. 2023d. Data management for large language models: A survey. *CoRR*, abs/2312.01700.

Jason Wei, Maarten Bosma, Vincent Y. Zhao, Kelvin Guu, Adams Wei Yu, Brian Lester, Nan Du, Andrew M. Dai, and Quoc V. Le. 2022. Finetuned language models are zero-shot learners. In *The Tenth International Conference on Learning Representations, ICLR 2022, Virtual Event, April 25-29, 2022*. OpenReview.net.

Shengguang Wu, Keming Lu, Benfeng Xu, Junyang Lin, Qi Su, and Chang Zhou. 2023. Self-evolved diverse data sampling for efficient instruction tuning. *CoRR*, abs/2311.08182.

Can Xu, Qingfeng Sun, Kai Zheng, Xiubo Geng, Pu Zhao, Jiazhan Feng, Chongyang Tao, and Daxin Jiang. 2023. Wizardlm: Empowering large language models to follow complex instructions. *CoRR*, abs/2304.12244.

Yunzhi Yao, Peng Wang, Bozhong Tian, Siyuan Cheng, Zhoubo Li, Shumin Deng, Huajun Chen, and Ningyu Zhang. 2023. Editing large language models: Problems, methods, and opportunities. *CoRR*, abs/2305.13172.

Aohan Zeng, Xiao Liu, Zhengxiao Du, Zihan Wang, Hanyu Lai, Ming Ding, Zhuoyi Yang, Yifan Xu, Wendi Zheng, Xiao Xia, Weng Lam Tam, Zixuan Ma, Yufei Xue, Jidong Zhai, Wenguang Chen, Zhiyuan Liu, Peng Zhang, Yuxiao Dong, and Jie Tang. 2023. GLM-130B: an open bilingual pre-trained model. In *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net.

Ningyu Zhang, Yunzhi Yao, Bozhong Tian, Peng Wang, Shumin Deng, Mengru Wang, Zekun Xi, Shengyu Mao, Jintian Zhang, Yuansheng Ni, Siyuan Cheng, Ziwen Xu, Xin Xu, Jia-Chen Gu, Yong Jiang, Pengjun Xie, Fei Huang, Lei Liang, Zhiqiang Zhang, Xiaowei Zhu, Jun Zhou, and Huajun Chen. 2024. A comprehensive study of knowledge editing for large language models. *CoRR*, abs/2401.01286.

Shengyu Zhang, Linfeng Dong, Xiaoya Li, Sen Zhang, Xiaofei Sun, Shuhe Wang, Jiwei Li, Runyi Hu, Tianwei Zhang, Fei Wu, and Guoyin Wang. 2023. Instruction tuning for large language models: A survey. *CoRR*, abs/2308.10792.

Wayne Xin Zhao, Kun Zhou, Junyi Li, Tianyi Tang, Xiaolei Wang, Yupeng Hou, Yingqian Min, Beichen Zhang, Junjie Zhang, Zican Dong, Yifan Du, Chen Yang, Yushuo Chen, Zhipeng Chen, Jinhao Jiang, Ruiyang Ren, Yifan Li, Xinyu Tang, Zikang Liu, Peiyu Liu, Jian-Yun Nie, and Ji-Rong Wen. 2023. A survey of large language models. *CoRR*, abs/2303.18223.

Chunting Zhou, Pengfei Liu, Puxin Xu, Srini Iyer, Jiao Sun, Yuning Mao, Xuezhe Ma, Avia Efrat, Ping Yu, Lili Yu, Susan Zhang, Gargi Ghosh, Mike Lewis, Luke Zettlemoyer, and Omer Levy. 2023. LIMA: less is more for alignment. *CoRR*, abs/2305.11206.

A Appendix

A.1 Installation

Currently, EasyInstruct offers three installation options, each accompanied by its corresponding installation script. Users can choose the option that best suits their specific requirements.

A.1.1 Installation from GitHub Repository

The first option is to install the latest version of EasyInstruct from the GitHub repository. The installation script is shown in Figure 5.

A.1.2 Installation for Local Development

The second option is to download the source code for local development. The installation script is shown in Figure 6.

A.1.3 Installation from PyPI

The third option is to install the package from The Python Package Index (PyPI), which may not be the latest version but still supports most of the features. The installation script is shown in Figure 7.

A.2 Quick-start

We provide two ways for users to quickly get started with EasyInstruct. Users can either use the shell script or the Gradio app based on their specific needs.

A.2.1 Shell Script

Step1: Prepare a configuration file. Users can easily configure the parameters of EasyInstruct in a YAML-style file or just quickly use the default parameters in the configuration files we provide. Figure 8 is an example of the configuration file for Self-Instruct.

Step2: Run the shell script. Users should first specify the configuration file and provide their own OpenAI API key. Then, run the following shell script in Figure 10 to launch the instruction generation or selection process.

A.2.2 Gradio App

We provide a Gradio app for users to quickly get started with EasyInstruct. Users can choose to launch the Gradio App locally on their own machines or alternatively, they can also try the hosted Gradio App[9] that we provide on HuggingFace Spaces.

[9] https://huggingface.co/spaces/zjunlp/EasyInstruct.

A.3 Detailed Hyper-Parameters

See Table 3.

Name	LLaMA-2-7b
batch_size	256
micro_batch_size	8
epochs	3
learning rate	3e-4
cutoff_len	512
val_set_size	1,000
lora_r	16
lora_alpha	32
lora_dropout	0.05

Table 3: Detailed hyper-parameters we use in experiments.

A.4 Prompt Template for Instruction Tuning

For both training and evaluation, we utilize the same prompt templates used by Alpaca-LoRA, shown in Table 4.

Prompt Template for Instruction Tuning
Prompt with Input:
Below is an instruction that describes a task, paired with an input that provides further context. Write a response that appropriately completes the request. ### Instruction: **{instruction}** ### Input: **{input}** ### Response:
Prompt without Input:
Below is an instruction that describes a task. Write a response that appropriately completes the request. ### Instruction: **{instruction}** ### Response:

Table 4: Prompt Template for instruction tuning.

A.5 API Services Available in EasyInstruct

Table 5 lists a range of API service providers and their corresponding LLM products that are currently available in EasyInstruct.

```
pip install git+https://github.com/zjunlp/EasyInstruct@main
```

Figure 5: Installation script from Github repository.

Model	Description	Default Version
OpenAI		
GPT-3.5	A set of models that improve on GPT-3 and can understand as well as generate natural language or code.	`gpt-3.5-turbo`
GPT-4	A set of models that improve on GPT-3.5 and can understand as well as generate natural language or code.	`gpt-4`
Anthropic		
Claude	A next-generation AI assistant based on Anthropic's research into training helpful, honest, and harmless AI systems.	`claude-2`
Claude-Instant	A lighter, less expensive, and much faster option than Claude.	`claude-instant-1`
Cohere		
Command	An instruction-following conversational model that performs language tasks with high quality, more reliably, and with a longer context than cohere's base generative models.	`command`
Command-Light	A smaller, faster version of Command. Almost as capable, but a lot faster.	`command-light`

Table 5: API service providers and their corresponding LLM products that are currently available in **EasyInstruct**.

```
git clone
↪  https://github.com/zjunlp/EasyInstruct
cd EasyInstruct
pip install -e .
```

Figure 6: Installation script for local development.

```
pip install easyinstruct
```

Figure 7: Installation script using PyPI.

```
generator:
  SelfInstructGenerator:
    target_dir: data/generations/
    data_format: alpaca
    seed_tasks_path:
↪   data/seed_tasks.jsonl
    generated_instructions_path:
↪   generated_instructions.jsonl
    generated_instances_path:
↪   generated_instances.jsonl
    num_instructions_to_generate: 100
    engine: gpt-3.5-turbo
    num_prompt_instructions: 8
```

Figure 8: Example configuration file of `Generators`.

```
selector:
  source_file_path:
  target_dir: data/selections/
  target_file_name: case.jsonl
  LengthSelector:
    min_instruction_length: 3
    max_instruction_length: 150
    min_response_length: 1
    max_response_length: 350
  Deduplicator:
  RougeSelector:
    threshold: 0.7
  GPTScoreSelector:
    engine: gpt-3.5-turbo
    threshold: 4
  MTLDSelector:
    ttr_threshold: 0.72
    min_mtld: 8
    max_mtld: 22
  PPLSelector:
    threshold: 200
    model_name: gpt2
    device: cuda
  RandomSelector:
    num_instructions_to_sample: 100
    seed: 42
```

Figure 9: Example configuration file of `Selectors`.

```
config_file=""
openai_api_key=""

python demo/run.py \
    --config  $config_file\
    --openai_api_key $openai_api_key \
```

Figure 10: Shell script for quick-start of EasyInstruct.

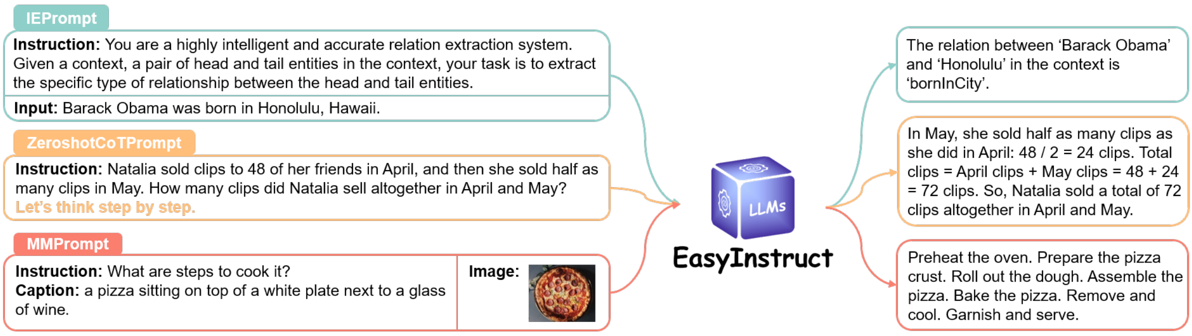


Figure 11: Example features in the `Prompts` module, including Information Extraction, Chain-of-Thought Reasoning, and Multimodal Prompting.

A.6 Example features in the `Prompts` module

A.7 Acknowledgements

We thank the developers of the self-instruct[10] library for their significant contributions to the NLP community. We thank the LLaMA team for providing us access to the models, and open-source projects, including Alpaca[11], Alpaca-LoRA[12] and AlpacaEval[13].

[10] https://github.com/yizhongw/self-instruct
[11] https://github.com/tatsu-lab/stanford_alpaca
[12] https://github.com/tloen/alpaca-lora
[13] https://github.com/tatsu-lab/alpaca_eval