ML-Bench: Large Language Models Leverage Open-source Libraries for Machine Learning Tasks

Anonymous ACL submission

Abstract

Large language models have shown promising performance in code generation benchmarks. However, a considerable divide exists between these benchmark achievements and their practical applicability, primarily attributed to real-world programming's reliance on pre-existing libraries. Instead of evaluating LLMs to code from scratch, this work aims to propose a new evaluation setup where LLMs use open-source libraries to finish machine learning tasks. Therefore, we propose ML-BENCH, a benchmark to evaluate how well these models use open-source libraries for machine learning tasks. It includes 10,100 samples spanning 169 tasks from 18 GitHub repositories, requiring models to understand complex documents and code structures. Notably, while GPT-4 exhibits remarkable improvement over other LLMs, it completes only 33.82% of tasks. Furthermore, we present an agent baseline ML-AGENT, which navigates codebases and generates executable code effectively.

1 Introduction

Large Language Models (LLMs) have demonstrated remarkable capabilities in function-level code generation, producing sophisticated code snippets with remarkable performance (Li et al., 2022; Austin et al., 2021b; Hendrycks et al., 2021; Chen et al., 2021a) in existing benchmarks.

Despite these significant strides, a notable gap remains between the competencies exhibited by these models under the dynamic requirements of real-world programming scenarios (Jimenez et al., 2023). Existing code generation benchmarks (Chen et al., 2021a; Austin et al., 2021a) evaluate LLM's ability to write code from scratch, while real-world programming often employs pre-existing and open-source libraries to accomplish tasks effectively (Zan et al., 2023) as Fig. 1. These

Human Eval

Instruction:

Return list of all prefixes from shortest to longest of the input string.

Ground Truth Code:

```
def all_prefixes(string: str) -> List[str]:
    result = []
    for i in range(len(string)):
        result.append(string[:i+1])
    return result
```

ML-Bench

Instruction:

Use the DualGAN model with facades dataset , learning rate to be set to 0.0001, the batch size to be 16, the number of training epochs to be 100 and the image size to be set to 128.

Ground Truth Code:

```
python3 dualgan.py --dataset_name facades --lr
0.0001 --n_epochs 100 --batch_size 16 --img_size 128
```

ML-Bench

Instruction:

Write a code for text classification using BERT.

Ground Truth Code:

```
from transformers import BertTokenizer
import torch
tokenizer = BertTokenizer.from_pretrained ('bert-base')
text = "Example text for classification"
inputs = tokenizer(text, padding=True, truncation=True)
.....
```

Figure 1: Examples of ML-Bench compared with existing code benchmark HumanEval. ML-Bench focuses on using open-source libraries to finish tasks.

open-source libraries offer robust, battle-tested solutions to various problems. Therefore, measuring the effectiveness of code LLMs should not merely be restricted to function generation (Shrivastava et al., 2023; Liu et al., 2023; Zhang et al., 2023). It should also include their ability to utilize open-source libraries and accurately configure parame-

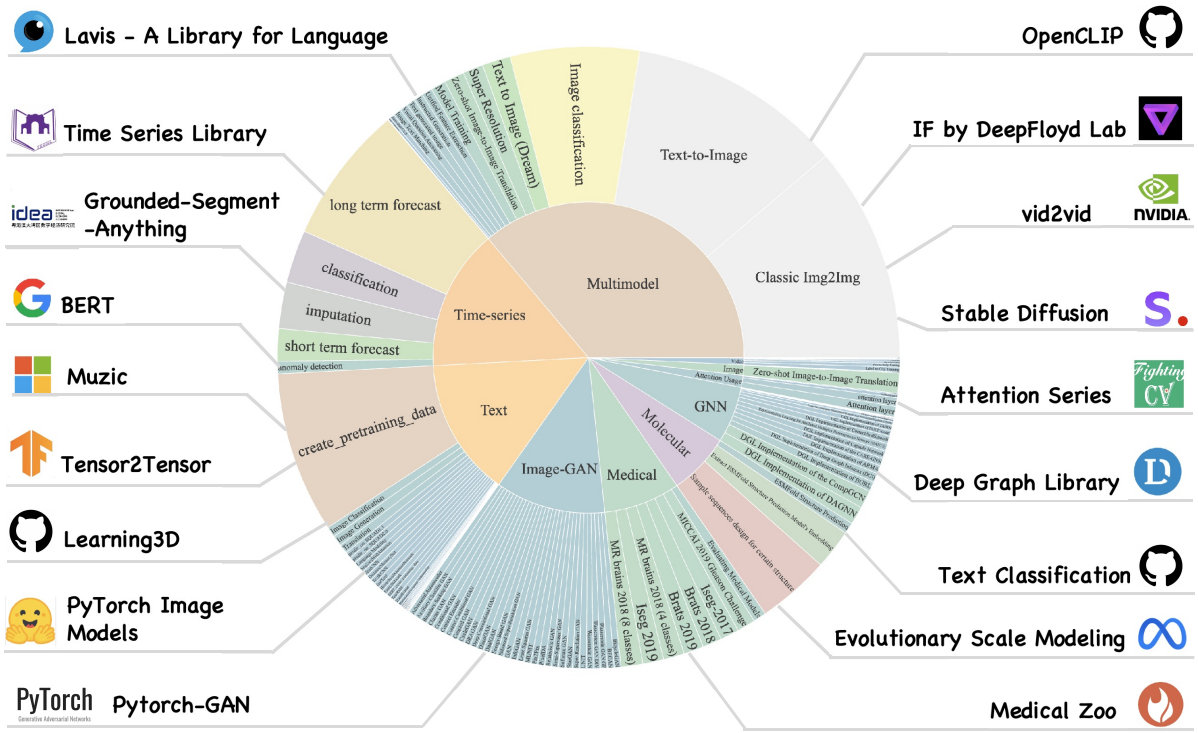

Figure 2: ML-BENCH encompasses **18 prominent GitHub repositories** and aggregates a total of **10,100 samples**.

ters when executing programming files.

This target often demands not just code generation but also comprehension of natural language documentation appended to the code, such as README files. README files typically provide a walkthrough of the open-source library, offering detailed narratives on task examples that the library can handle, including guidance on parameter selection. In reality, the parameters stipulated by the user may not always align with those enumerated in the README examples, mandating the LLM to accurately adapt the parameter values based on the user's specific requirements. Hence, an LLM with the ability to effectively interpret and leverage the information within README files and precisely customize parameters in code based on README examples would significantly advance machine learning automation.

Based on the aforementioned motivation, we introduce ML-BENCH, a comprehensive and realistic benchmark dataset designed to evaluate LLMs in understanding user instructions, navigating codebases, and writing executable code. ML-BENCH provides high-quality instructions along with corresponding executable ground truth code that meets the requirements specified in the instructions. ML-BENCH consists of 10100 samples spanning 169 tasks over 18 notable machine learning GitHub repositories as Fig. 2. We carefully provide various settings to accommodate different LLMs. Additionally, we carefully design comprehensive and fair evaluation metrics (i.e., Pass@k conditioned by additional checks) to quantify the performance.

ML-BENCH proposes new challenges to LLMs. Empirically, we find that GPT-4 performs remarkable improvement over other LLMs but still only manages to accomplish less than 40% of the tasks. LLMs suffer from hallucinations and perform poorly. A key insight from our study is the crucial need for LLMs to comprehend long-text documentation, not just generate code. We also propose an agent baseline ML-AGENT, an autonomous language agent developed to fill the aforementioned gaps. The agent is able to understand natural language and instructions, navigate the codebase, and produce requisite code effectively.

2 Dataset and Benchmark

In this section, we provide information about ML-BENCH, including definition of the task (Sec 2.1), how to select repositories (Sec 2.2), annotation pipeline (Sec 2.3), and dataset statistics (Sec 2.4).

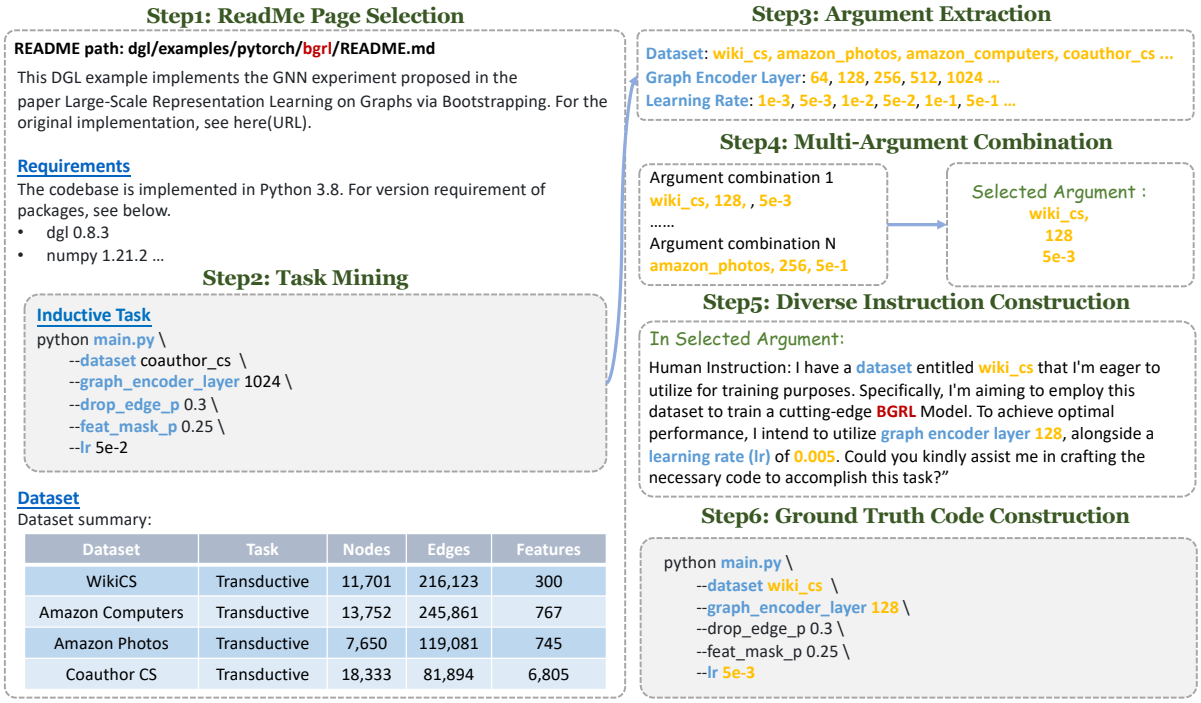

Figure 3: The construction pipeline of the ML-BENCH. **Step 1**: Find the readme file in the repository that implements a specific task. **Step 2**: Extract essential task-related information. **Step 3**: Extract parameters from task mining. **Step 4**: Leverage the details extracted from the readme file and multiple combinations of parameters are formulated. **Step 5**: Create the instructions for each parameter combination. **Step 6**: Substitute the code or script within the readme, serving as the ground truth code for ML-BENCH.

2.1 Task Formulation

In our benchmark, we focus on a scenario where, given a GitHub repository $\mathcal{F}$, the language model has access to all files $f \in \mathcal{F}$ within the repository. Upon receiving a user instruction i with a set of specific parameters $\arg_i \in \mathcal{A}$. We emphasize that the generated code c must align with the user's instruction i, particularly in terms of all specified parameters $\arg_i \in \mathcal{A}$, and must be executable.

2.2 Repositries Selection

We collected papers with corresponding GitHub links from the PapersWithCode website[1], then organized these papers based on their categorical information. From each category, we selected the top repositories based on the ranking of stars. Further, through manual filtering, we identified the most representative and feature-rich repositories from this subset, making our large group of 18 repositories. More details about the selected repositories can be found in Sec. 2.4 and Appendix B.

2.3 Annotation Pipeline

Regarding data annotation, relying solely on human efforts can be time and labor-intensive. To address this challenge, we have developed a comprehensive workflow with the help of LLMs to facilitate the annotation process. Some of the authors are employed as annotators. Our annotators consist of graduate students with proficient programming skills. Fig. 3 shows The construction pipeline of the ML-BENCH. Each GitHub repository is assigned to a designated annotator. [2]

README Page Selection: When users utilize a GitHub repository to accomplish tasks, the README file is often the most valuable resource. Therefore, we instruct annotators to carefully review the contents of the repository first. Their primary task includes identifying all README files within the repository. This encompasses not only the main README but also those located in various sub-directories, which describe the contents of their respective folders. On average, each GitHub covers 12 README pages, while GNN covers 154 README pages.

Task Mining: After identifying all the README

[1] https://paperswithcode.com/

[2] The minimum hourly wage in the United States is near $7.5, which can be found at https://www.worker.gov/. On average, annotating an example takes 5 hours. Consequently, the cost per annotator amounts to $37.5.

files contained within a GitHub repository, our annotators are tasked with pinpointing each task covered by every README, along with the code scripts capable of accomplishing these tasks. The annotators are instructed to select the most representative and practical tasks inside each GitHub. On average, each GitHub repository covers 9 tasks.

Parameters Extraction: In addition to specifying the task, users may also need to define important parameters, as the setting of these parameters is crucial to the effectiveness of finishing the task. Annotators all have experience in machine learning the annotators are tasked with locating key parameters in the code (such as batch size, epochs, model architecture, and dataset). Our target is to find representative parameters that are likely to be searched and tested during practical experiments.

Multi-Parameter Combination: Once the code and its pertinent parameters for each task were identified, the annotators iteratively and systematically combined candidate values for selected parameters. This approach resulted in numerous combinations of parameters relevant to each task.

Diverse Instruction Construction: For each set of parameters obtained during the Multi-Parameter Combination process, we utilized ChatGPT to generate instructions specifically tailored to the task and these parameters. We meticulously design templates to ensure that output instructions contain the given parameter requirements and exhibit a strong diversity by providing specific parameter indications and various diverse examples.

Ground Truth Code Construction: Based on the parameter combinations obtained during the Multi-Parameter Combination process, we further produce a code template based on the code for the target task. Then we use the code template to construct ground truth code.

Quality Test: We have conducted rigorous quality checks on the dataset. Specifically, for the ground truth output, we perform strict checks for executability and parameters. Any data that did not meet the requirements were discarded. Regarding the instructions, we ensured that all parameters were explicitly specified within the instructions.

2.4 Overall Statistics

Referring to the data construction process detailed in Section 2.3, we carefully curated a dataset tailored to prominent Machine Learning domains, serving as the cornerstone of our ML-BENCH. This extensive dataset is partitioned into separate training and testing subsets, designed for distinct purposes in training and evaluation. Tab.7 delineates the specific data statistics. Further detailed data metrics for each repository utilized can be found in Appendix C.

The comprehensive statistics of ML-BENCH are itemized in Appendix E, which is culled from our selected GitHub repositories, offering statistical insights into the diverse range of instructions we meticulously developed.

3 Proposed Agent Baseline: ML-AGENT

In our practice, an autoregressive model can not solve the problems in ML-BENCH in a single step without user intervention. So we introduce ML-AGENT: an agent baseline to inspire the work performed on ML-BENCH. Our objective is to emulate humans in the situation when given an ML task in which they go through the following steps: Firstly, humans typically prefer to retrieve task-specific tools from the Web instead of creating code from scratch. After preliminary retrieval, they will select the prime ones and attempt to utilize the tools for their tasks. Then, humans will look for helpful content in tool documents. To ensure proper utilization of the tools, they may find it necessary to scan the parameter definitions within the files. Besides, they also decide whether the tool is useful when finding details. Once the tools are confirmed to be correct, humans will modify the tool's code according to their requirements.

To this end, we structured the agent design into four distinct phases, encompassing the processes of retrieval, ranking, planning, and generating. We use the function-call ability of GPT-4 to homogenize the communication of modules in the agent. Now we introduce the features of our ML-AGENT:

Retrieval: ML-AGENT extracts keywords from the user's instructions, including factors like tasks and desired models. These keywords are then used to retrieve the top-k related repositories on GitHub.

Ranking: The agent ranks the repositories based on the relevance of their meta information, like repository descriptions, to the users' instructions.

Planning: The agent delves into the repositories. For each repository, ML-AGENT extracts the directory structure, scans the files in the repositories, decides, and extracts related README contents and code snippets. Providing a feasible way to generate code to meet the user's requirements.

Generating: Given the selected README contents and code snippets, ML-AGENT will generate corresponding code or script or refine to the previous planning step; the actions form a loop, which can only be exited by either writing or refining. We show the structure of ML-AGENT in App. J.

4 Experiments

This section presents details about the conducted experiment. Including experimental setup (Sec. 4.1) and evaluation metrics (Sec. 4.2).

4.1 Experimental Setup

For single models, we have three experimental setups, differing in how the content of README files in the repositories is presented:

Oracle Segment: The annotators were asked to record the corresponding segments in the README files when writing the ground-truth code. An oracle segment usually comprises a task-related code and its natural language explanation.

Raw README: In this setting, we present the entirety of all the README files to the model without any modifications. The texts will be truncated when the length of the README files exceeds the model's maximum context limit.

BM25 Retrieval: A BM25 retriever is used to assist the model in locating relevant information. We set a retrieval span of 10 sentences for the BM25 retriever using the NLTK sentence tokenizer [3], as the average length of an Oracle Segment is 9.5.

We specifically sampled 260 examples from 14 repositories as our test set. We introduce two setups to fine-tune open-source models: the ID (In-Distribution) setting and the OOD (Out-of-Distribution) setting. The remaining samples from the 14 repositories served as training set for the ID setting. The train set consists of all the samples in the other four repositories for the OOD setting.

Model input includes their specific system prompt, human instruction, and the chosen README content, as shown in Appendix K. To ensure we did not exceed the tokens limitation for closed-source models, we utilized GPT-3.5-16k, GPT-4-32k, and Claude 2. We chose CodeLlama-7b-Instruct, DeepSeek-Coder-6.7b-base, and Llama-2-7b-hf [4] to test the performance of open-source models. Since the first two models have garnered significant attention in code generation. LLaMA is directly compared with CodeLlama to assess whether the training of code data significantly improves the model performance on ML-BENCH.

For agents, We chose AutoGen [5] equipped with GPT-4 as a comparative agent for our task. The chunk token size utilized was set to 2000. For ML-AGENT, we provide only human instruction and all the repositories in ML-BENCH test set as retrieval candidates; for AutoGen, we provide human instruction and the README files of the specific repository, as shown in Appendix K.

4.2 Evaluation Metrics

we employ pass@K to assess the output generated by models or agents. Pass@K refers to the probability that the correctly executed code is generated, conditioned by hitting the claimed parameters and the same generation type as the reference content.

5 Result Analysis

This section discusses data leakage, shows our experimental results, and analyzes the performance of models and agents conducted on ML-BENCH.

Data leakage. Our early experiments found that models tended to generate code regardless of whether the content the model was supplied with was about script or code. This is related to the fact that model trainers expect they can get high scores on other code benchmarks, causing these polished repositories to be more likely to be put into the training set compared to lesser-known ones, and code data is a larger percentage than script data.

To mitigate the influence of data leakage, the type and parameters of the generated result are checked to see whether it is the same as those present in the provided documents and user instructions before execution. We show the updating status for all repositories in Appendix B and the cutoff date of training data for models in Appendix I.

Main result. Tab. 1 and Tab. 2 show the Pass@K results for models and agents. Across the table, we observe that:

1. GPT-4 consistently demonstrates superior performance. However, ML-BENCH is still challenging for GPT-4, as it achieved a Pass@5 score under 50%. Meanwhile, LLaMA almost does not solve any problem.

[3] https://www.nltk.org/api/nltk.tokenize

[4] Due to hardware limitations, the input of open-source models is truncated to the first 8k tokens for training and inference.

[5] our implementation of AutoGen

Models	Oracle Segment			Raw README			BM25 Retrieval		
	Pass@1	**Pass@2**	**Pass@5**	**Pass@1**	**Pass@2**	**Pass@5**	**Pass@1**	**Pass@2**	**Pass@5**
GPT-4 *	**33.82**	**39.71**	**48.53**	**30.88**	**36.76**	**45.59**	**22.06**	**23.53**	**27.94**
GPT-3.5 *	27.94	33.82	38.23	15.07	20.55	30.14	13.70	19.18	24.66
Claude 2 *	21.92	32.88	34.25	27.40	34.25	35.62	9.59	13.70	20.55
GPT-3.5	21.92	27.31	36.92	13.46	20.38	35.39	10.00	15.00	22.69
Claude 2	18.46	26.54	30.38	25.38	30.77	32.31	8.08	10.77	16.92
CodeLlama-raw	8.85	11.92	21.15	1.54	3.85	8.85	0.77	2.69	8.85
CodeLlama-ID	17.69	23.46	30.77	/	/	/	2.69	6.15	9.62
CodeLlama-OOD	15.76	21.53	28.46	/	/	/	1.92	3.46	5.38
DeepSeekCoder-raw	1.54	2.31	7.31	0.00	0.00	1.36	0.00	1.54	2.31
DeepSeekCoder-ID	18.46	25.38	30.38	/	/	/	7.69	10.32	13.00
DeepSeekCoder-OOD	0.77	2.31	8.85	/	/	/	2.69	5.00	6.54
LLaMA-raw	0.00	0.00	0.77	0.00	0.00	0.00	0.00	0.00	0.00
LLaMA-ID	0.38	3.46	6.54	/	/	/	1.15	2.69	3.08
LLaMA-OOD	0.00	0.00	0.38	/	/	/	0.77	1.15	2.31

Table 1: Pass@K (%) results. Pass@1, Pass@2, and Pass@5 scores are reported for models. We report the results of GPT-3.5 and Claude 2 on the quarter set to show the distribution bias among the full and quarter test sets. We didn't fine-tune open-source models under the Raw README setting. * Means evaluated on quarter set in ML-BENCH bound to the budget.

2. After training, open-source models are still less effective than the worst closed-source model. The observation that CodeLlama-raw and DeepSeekCoder-raw beat LLaMA-raw shows that training on code data improves script skills significantly.
3. Providing accurate and short task-related content helps improve all models' performance. However, inflexible retrieval methods such as BM25 have side effects on the model's ability to solve problems in ML-BENCH. We show the BLEU scores between Oracle Segments and BM25 retrieved contents in Appendix I.
4. ML-AGENT achieved noteworthy results without any provided repository information. We observed that ML-AGENT sometimes delves the Python files in cases where the README files should be consulted, leading it to refer to the code. However, to keep the evaluation metric consistent, we treat these results as hallucination errors when performing the type check. The same as AutoGen.

Error defination. By analyzing the execution log, we find that the errors for models and agents in ML-BENCH fall into four categories:

Hallucination Errors: These errors include instances when the models misinterpreted the user's intention, misplaced Python Code and Bash Script, or generated random or irrelevant code.

Agent Name	**Pass@1**	**Pass@2**	**Pass@5**
ML-AGENT *	23.53	35.29	47.06
AutoGen*	6.06	6.06	6.06

Table 2: Pass@K results for agents

Lack of Knowledge or Information: This type of error primarily stems from the model's inability to fulfill user requirements based on crucial parameters. Possible types of errors are as follows:

1. Code inner information. The models sometimes lack sufficient information necessary to satisfy the user's requirements. For instance, this deficiency might involve missing parameter names (`-lr` and `-learning rate`) or unavailable options (it only accepts 'Citeseer' when the input given was 'citeseer').
2. Domain knowledge. The models sometimes lack the domain-specific knowledge required to handle certain instances. For example, in BERT, the models simultaneously generated `-case=True` and `-model=uncased`.
3. Grammar knowledge. This happens when the models incorrectly identify and handle certain command line symbols. Like the $ symbol, which could affect execution.
4. Local data information. The models were not supplied with enough crucial parameter information for specific commands, leading to the inability to find paths and successful execution. While less common, this error was particularly notable in the OpenCLIP.

Knowledge Manipulation: Take BERT, for instance, where the model needed to integrate `DIR=/model/` and `-model_dir=$DIR` to form

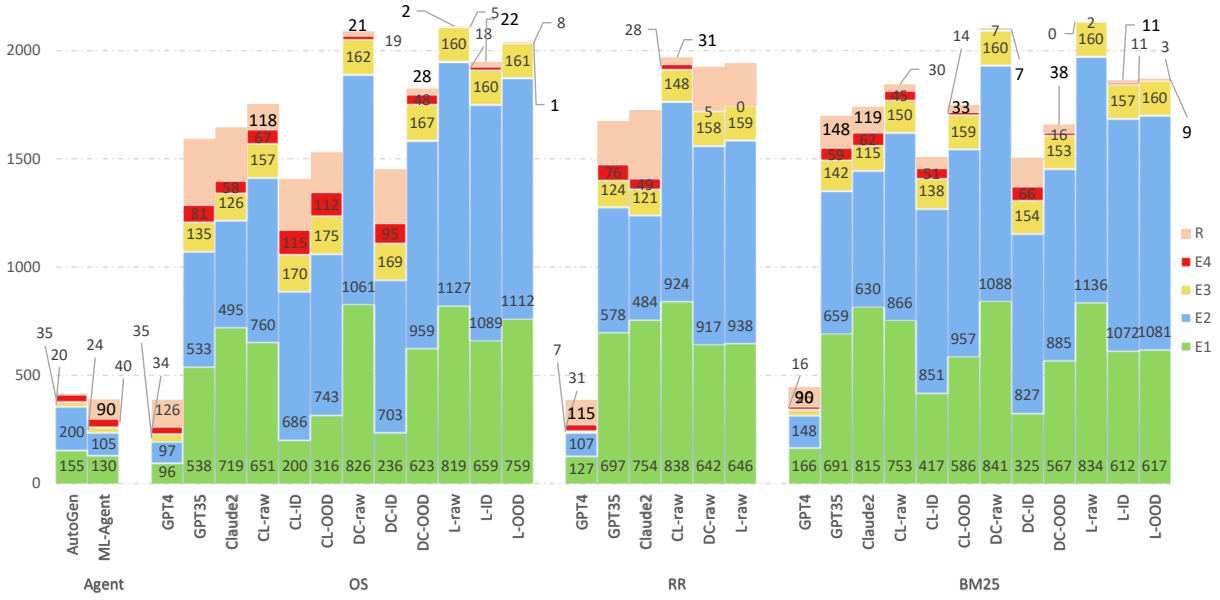

Figure 4: Quantification of models and settings errors for five times attempts. The total statistic results are 1300 for the full test set and 340 for the quarter test set. Statistical results that exceed these numbers are caused by multiple errors made on one result simultaneously. For models, CL means CodeLlama, DC means deepseek-coder, and L means LLaMA. For errors, E1 means Hallucination Errors, E1 means Lack of Knowledge or Information, E3 means Knowledge Manipulation, E4 means Syntax Errors, and R means right results.

`–model_dir=/model`. There were also cases where it couldn't merge `/model_name in /data` into a proper path format like `/data/model_name`. Similar incidents were observed in OpenCLIP.

Syntax Errors: These errors cover instances of incorrect code generation of syntax errors instead of hallucination, mainly Python syntax errors such as the use of undefined variables. These errors arise from cases that prefer generating Python code.

Settings and models error analysis. Fig. 4 overviews the error statistics for models and settings. Generally, the proportions of E3 and E4 are small. For E3, this is because the capable cases of producing such errors constitute a small portion. For E4, only after passing through the type check and parameters check does the generation result undergo execution, and execution exposes E4.

For closed-source models, we find that Claude 2 is more prone to hallucinations than GPT-3.5. However, its ability to fulfill user-stated requirements (represented by E2) is better than that of GPT-3.5. Compared to the RR setting, under the BM25 setting, neither GPT-3.5 nor Claude 2 exhibits an increase in hallucination but an increase in the proportion of E2. For GPT-4, both the occurrence of E1 and E2 have increased. Logs show this phenomenon occurs because GPT-4 generates code without considering the content in cases involving task-irrelevant information, but not GPT-3.5 and Claude 2. Compared to the Raw README setting, oracle segment provision leads to a decrease in the quantities of E1 and E2, while the differences in E3 and E4 are insignificant. This suggests that the closed-source models' knowledge manipulation and Python code generation abilities are not significantly affected by whether an oracle segment is provided. We tend to attribute these to the models' inherent ability rather than the reference.

For open-source models, we find that without supervised fine-tuning, the models make errors of E1 and E2 with nearly equal frequency, and fine-tuning can significantly reduce the percentage of hallucination errors under both OOD and ID settings. In addition, both Codellama and Deepseek have shown significant improvement compared to LLaMA after fine-tuning on ML-BENCH.

For agents, ML-AGENT made more hallucinatory mistakes than GPT-4. This is mainly because the agent did not always stay within the file type consistent with the predefined output type in ML-BENCH during the last two steps. This aspect is demonstrated by the significant improvement in the ML-AGENT's pass@K performance as K increases in Tab. 2. Indeed, multiple attempts aid ML-AGENT in locating content types consistent with the ML-BENCH's requirements. This underscores the capability that such agents need to enhance. For AutoGen, a similar situation persists, with the extensive presence of E2 within the gen-

Repos	ML-AGENT *	GPT 4*			Claude 2*			GPT 3.5			Claude 2		
		OS	RR	BM25	OS	RR	BM25	OS	RR	BM25	OS	RR	BM25
DGL	**80.00**	**80.00**	60.00	60.00	40.00	20.00	80.00	**47.62**	23.81	23.81	28.57	19.05	14.29
BERT	50.00	50.00	50.00	16.67	0.00	**80.00**	16.67	**22.73**	13.63	13.63	0.00	4.54	0.00
Lavis	57.14	42.86	71.43	42.86	57.14	**85.71**	14.29	55.56	**70.37**	51.85	51.85	59.26	29.63
If	**100.00**	**100.00**	**100.00**	33.33	**100.00**	0.00	13.33	71.43	61.90	52.38	71.43	**76.19**	52.38
vid2vid	0.00	50.00	**75.00**	50.00	0.00	25.00	50.00	**92.31**	76.92	69.23	76.92	38.46	15.38
ESM	0.00	60.00	0.00	80.00	0.00	**100.00**	20.00	47.06	29.41	**58.82**	5.88	11.76	11.76
OpenCLIP	**66.67**	**66.67**	**66.67**	**66.67**	**66.67**	**66.67**	0.00	**63.63**	36.36	54.55	**63.63**	**63.63**	45.46
TSL	**75.00**	25.00	25.00	0.00	25.00	0.00	0.00	**14.29**	**14.29**	0.00	7.14	7.14	0.00
EAP	80.00	**100.00**	80.00	0.00	100.00	20.00	80.00	66.66	70.83	33.33	70.83	**83.33**	20.83
Py-GAN	**12.50**	0.00	**12.50**	0.00	0.00	**12.50**	0.00	**6.67**	0.00	0.00	0.00	0.00	0.00
Py-IM	0.00	0.00	0.00	0.00	0.00	0.00	0.00	**20.00**	0.00	0.00	0.00	0.00	0.00
Learning3d	0.00	0.00	0.00	0.00	**25.00**	0.00	**25.00**	23.53	**47.06**	35.29	17.65	0.00	0.00
muzic	40.00	**80.00**	60.00	40.00	60.00	20.00	20.00	66.67	**72.22**	61.11	38.89	33.33	33.33
Grounded-SAM	**60.00**	**60.00**	**60.00**	20.00	0.00	0.00	0.00	0.00	**20.00**	0.00	5.00	35.00	10.00
Total	47.06	**48.53**	45.59	27.94	34.25	35.61	20.55	36.92	35.39	22.69	30.38	32.31	16.92

Table 3: This table shows the Pass@5 scores of ML-AGENT, GPT-4, and Claude 2 on Oracle Segment, Raw README, and BM25 Retrieval settings in different GitHub repositories on quarter set.

erated code contributing to its overall low scoring.

Repositories analysis. We report Pass@5 scores of models in Tab. 3 and the detailed GPT-4 and ML-AGENT results in Appendix G for each repository. Shows that the impact of offering an Oracle segment is notable for repositories that prefer Bash Script in the folder READMEs, such as DGL. Also, for DGL, ML-AGENT sometimes generates Python code by viewing Python files but not extracting content from README, which leads to even correct results being classified as E1.

When information is scattered across sections of a README, the Oracle Segment setting performs worse than the Raw README setting. A good example is Lavis. The task of If looks more accessible because it is centralized with less redundant information in the README. However, Learning3d, TSL, and Py-IM seem complicated for models. This may caused by their insufficient and decentralized information in README and the proportion of code. We show their features in Appendix F.

The scores of ML-AGENT on DGL, TSL, and Lavis show that the agent can retrieve and integrate information accurately, demonstrating it is a strong baseline for ML-BENCH. However, its performance in simpler tasks such as ESM, vid2vid, and muzic suggests that there is still room for improvement.

6 Related work

Code generation has been a subject of longstanding inquiry in the field of NLP. It demonstrates activity in the formulation of methodologies and the establishment of benchmarks (Chen et al., 2021a; Christopoulou et al., 2022; Cassano et al., 2022; Orlanski et al., 2023; Wang et al., 2023d; Tang et al., 2023a,b). While existing code benchmarks target advancing the capability for function-level code generation, ML-BENCH focuses on leveraging code generation for facilitating tool usage. More details are shown at Tab. 4

The objective of function-level code generation is to produce code snippets meeting user requirements or contribute to code completion (Feng et al., 2020; Li et al., 2022), encompassing the development of foundational models for code generation (Zheng et al., 2023; AI, 2023).

Bench Name	Type	Domain	# Samples
ML-Bench	Func. Calling	ML	10,000
HumanEval	Func. Prod.	Python	164
MBPP	Func. Prod.	Python	1,000
DS-1k	Func. Prod.	ML	1,000

Table 4: The characteristics of code generation benchmarks. The compared benchmarks include HumanEval (Chen et al., 2021b), MBPP (Austin et al., 2021a) and DS-1k (Lai et al., 2023).

7 Conclusion

In this paper, we introduced ML-BENCH, a comprehensive benchmark designed to evaluate the effectiveness of utilizing existing packages for ML tasks. In addition, we proposed ML-AGENT, an agent baseline capable of reading files and writing scripts to fulfill user needs. We carefully provide various settings to accommodate different LLMs and carefully design comprehensive and fair evaluation metrics to quantify the performance. We believe that ML-BENCH contributes substantially to machine learning, offering researchers and practitioners new tools to advance the SOTA in automated machine learning processes.

limitation

Our study, while comprehensive within its scope, is subject to certain limitations that stem primarily from linguistic and data source constraints.

Linguistic Limitation - English as a Working Language We exclusively focused on English for our analyses and model development. This choice, while pragmatic due to English's prevalence in scientific literature and technical documentation, inherently limits the generalizability of our findings. English, as a language, possesses unique syntactic and semantic structures that may not be representative of other languages. Consequently, the applicability of our results to non-English contexts is uncertain. This linguistic limitation also restricts the diversity of perspectives and cultural nuances that non-English documents could offer.

Data Source Limitation - Reliance on GitHub Repositories in English Our reliance on GitHub repositories with documents exclusively in English introduces a selection bias. GitHub, while rich in open-source projects and documentation, may not comprehensively represent the broader landscape of software development practices and trends globally. This choice potentially overlooks significant contributions and insights from non-English-speaking communities. This limitation might impact the development of tools and models tailored to a more diverse set of programming environments and community needs.

Methodological Limitation - Relying on Pre-built Machine Learning Packages In our methodology, we utilized existing machine learning packages instead of developing algorithms from scratch. While this approach allowed us to leverage well-established, tested, and optimized tools, it also introduces certain constraints. Dependence on pre-built packages means our work is confined to the capabilities and limitations of these tools. This reliance could limit our ability to fully explore novel or unconventional approaches possible with custom-built algorithms. Moreover, this choice potentially impacts the reproducibility and customization of our findings. Researchers who seek to build upon our work may encounter similar constraints imposed by the pre-built packages we utilize. These limitations can hinder innovation and adaptation in different contexts or for specific usage.

Scope Limitation - Tasks Limited to README File Descriptions By strictly adhering to the specified tasks, our study may overlook potential applications or challenges not explicitly documented in README. This limitation can result in a narrower understanding of the tools we examined, as it fails to explore their full potential and applicability. The reliance on README descriptions also assumes that these documents comprehensively and accurately reflect all relevant aspects of the repositories, which may not always be accurate. Important tasks or nuances might be undocumented or underrepresented in these files.

Ethics Statement

In our work, we have carefully considered the ethical implications of our work, particularly in data annotation and related activities. Our methodologies and processes have been meticulously designed to ensure they are free from moral concerns. We affirm that our research practices, including data handling, have been conducted with the utmost integrity and in compliance with ethical standards.

Our approach has been guided by principles prioritizing respect for data integrity, transparency in our methods, and adherence to established ethical guidelines.

References

DeepSeek AI. 2023. DeepSeek Coder.

Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, et al. 2021a. Program synthesis with large language models. *arXiv preprint arXiv:2108.07732*.

Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie J. Cai, Michael Terry, Quoc V. Le, and Charles Sutton. 2021b. Program synthesis with large language models. *arXiv preprint arXiv:2108.07732*.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. 2020. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901.

Zefan Cai, Baobao Chang, and Wenjuan Han. 2023. Human-in-the-loop through chain-of-thought. *arXiv preprint arXiv:2306.07932*.

Federico Cassano, John Gouwar, Daniel Nguyen, Sydney Nguyen, Luna Phipps-Costin, Donald Pinckney, Ming-Ho Yee, Yangtian Zi, Carolyn Jane Anderson, Molly Q Feldman, Arjun Guha, Michael Greenberg, and Abhinav Jangda. 2022. MultiPL-E: A scalable and extensible approach to benchmarking neural code generation. *arXiv preprint arXiv:2208.08227*.

Liang Chen, Yichi Zhang, Shuhuai Ren, Haozhe Zhao, Zefan Cai, Yuchi Wang, Peiyi Wang, Tianyu Liu, and Baobao Chang. 2023a. Towards end-to-end embodied decision making via multi-modal large language model: Explorations with GPT4-Vision and beyond. *arXiv preprint arXiv:2310.02071*.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. 2021a. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. 2021b. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*.

Weize Chen, Yusheng Su, Jingwei Zuo, Cheng Yang, Chenfei Yuan, Chi-Min Chan, Heyang Yu, Yaxi Lu, Yi-Hsin Hung, Chen Qian, Yujia Qin, Xin Cong, Ruobing Xie, Zhiyuan Liu, Maosong Sun, and Jie Zhou. 2023b. AgentVerse: Facilitating multi-agent collaboration and exploring emergent behaviors. *arXiv preprint arXiv:2308.10848*.

Fenia Christopoulou, Gerasimos Lampouras, Milan Gritta, Guchun Zhang, Yinpeng Guo, Zhongqi Li, Qi Zhang, Meng Xiao, Bo Shen, Lin Li, et al. 2022. PanGu-Coder: Program synthesis with function-level language modeling. *arXiv preprint arXiv:2207.11280*.

Zhangyin Feng, Daya Guo, Duyu Tang, Nan Duan, Xiaocheng Feng, Ming Gong, Linjun Shou, Bing Qin, Ting Liu, Daxin Jiang, et al. 2020. Codebert: A pre-trained model for programming and natural languages. *arXiv preprint arXiv:2002.08155*.

Ran Gong, Qiuyuan Huang, Xiaojian Ma, Hoi Vo, Zane Durante, Yusuke Noda, Zilong Zheng, Song-Chun Zhu, Demetri Terzopoulos, Li Fei-Fei, and Jianfeng Gao. 2023. MindAgent: Emergent gaming interaction. *arXiv preprint arXiv:2309.09971*.

Significant Gravitas. 2023. AutoGPT.

Dan Hendrycks, Steven Basart, Saurav Kadavath, Mantas Mazeika, Akul Arora, Ethan Guo, Collin Burns, Samir Puranik, Horace He, Dawn Xiaodong Song, and Jacob Steinhardt. 2021. Measuring coding challenge competence with APPS. In *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track*.

Sirui Hong, Xiawu Zheng, Jonathan Chen, Yuheng Cheng, Ceyao Zhang, Zili Wang, Steven Ka Shing Yau, Zijuan Lin, Liyang Zhou, Chenyu Ran, et al. 2023. MetaGPT: Meta programming for a multi-agent collaborative framework. *arXiv preprint arXiv:2308.00352*.

Carlos E Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik Narasimhan. 2023. Swe-bench: Can language models resolve real-world github issues? *arXiv preprint arXiv:2310.06770*.

Yuhang Lai, Chengxi Li, Yiming Wang, Tianyi Zhang, Ruiqi Zhong, Luke Zettlemoyer, Wen-tau Yih, Daniel Fried, Sida Wang, and Tao Yu. 2023. Ds-1000: A natural and reliable benchmark for data science code generation. In *International Conference on Machine Learning*, pages 18319–18345. PMLR.

Guohao Li, Hasan Abed Al Kader Hammoud, Hani Itani, Dmitrii Khizbullin, and Bernard Ghanem. 2023. CAMEL: Communicative agents for "mind" exploration of large language model society. *arXiv preprint arXiv:2303.17760*.

Yujia Li, David Choi, Junyoung Chung, Nate Kushman, Julian Schrittwieser, Rémi Leblond, Tom Eccles, James Keeling, Felix Gimeno, Agustin Dal Lago, Thomas Hubert, Peter Choy, Cyprien de Masson d'Autume, Igor Babuschkin, Xinyun Chen, Po-Sen Huang, Johannes Welbl, Sven Gowal, Alexey Cherepanov, James Molloy, Daniel J. Mankowitz, Esme Sutherland Robson, Pushmeet Kohli, Nando de Freitas, Koray Kavukcuoglu, and Oriol Vinyals. 2022. Competition-level code generation with AlphaCode. *Science*, 378(6624):1092–1097.

Tianyang Liu, Canwen Xu, and Julian McAuley. 2023. RepoBench: Benchmarking repository-level code auto-completion systems. *arXiv preprint arXiv:2306.03091*.

Gabriel Orlanski, Kefan Xiao, Xavier Garcia, Jeffrey Hui, Joshua Howland, Jonathan Malmaud, Jacob Austin, Rishabh Singh, and Michele Catasta. 2023.

Measuring the impact of programming language distribution. In *Proceedings of the 40th International Conference on Machine Learning*.

Shishir G. Patil, Tianjun Zhang, Xin Wang, and Joseph E. Gonzalez. 2023. Gorilla: Large language model connected with massive APIs. *arXiv preprint arXiv:2305.15334*.

Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, Sihan Zhao, Lauren Hong, Runchu Tian, Ruobing Xie, Jie Zhou, Mark Gerstein, Dahai Li, Zhiyuan Liu, and Maosong Sun. 2023. ToolLLM: Facilitating large language models to master 16000+ real-world APIs. *arXiv preprint arXiv:2307.16789*.

Nan Shao, Zefan Cai, Chonghua Liao, Yanan Zheng, Zhilin Yang, et al. 2022. Compositional task representations for large language models. In *The Eleventh International Conference on Learning Representations*.

Yongliang Shen, Kaitao Song, Xu Tan, Dongsheng Li, Weiming Lu, and Yueting Zhuang. 2023. HuggingGPT: Solving AI tasks with chatGPT and its friends in hugging face. In *Thirty-seventh Conference on Neural Information Processing Systems*.

Disha Shrivastava, Denis Kocetkov, Harm de Vries, Dzmitry Bahdanau, and Torsten Scholak. 2023. RepoFusion: Training code models to understand your repository. *arXiv preprint arXiv:2306.10998*.

Xiangru Tang, Bill Qian, Rick Gao, Jiakang Chen, Xinyun Chen, and Mark B. Gerstein. 2023a. BioCoder: A benchmark for bioinformatics code generation with contextual pragmatic knowledge. *arXiv preprint arXiv:2308.16458*.

Xiangru Tang, Yiming Zong, Yilun Zhao, Arman Cohan, and Mark Gerstein. 2023b. Struc-Bench: Are large language models really good at generating complex structured data? *arXiv preprint arXiv:2309.08963*.

XAgent Team. 2023. XAgent: An autonomous agent for complex task solving.

Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. 2023a. Voyager: An open-ended embodied agent with large language models. In *NeurIPS 2023 Foundation Models for Decision Making Workshop*.

Lei Wang, Chen Ma, Xueyang Feng, Zeyu Zhang, Hao Yang, Jingsen Zhang, Zhiyuan Chen, Jiakai Tang, Xu Chen, Yankai Lin, Wayne Xin Zhao, Zhewei Wei, and Ji-Rong Wen. 2023b. A survey on large language model-based autonomous agents. *arXiv preprint arXiv:2308.11432*.

Peiyi Wang, Lei Li, Liang Chen, Dawei Zhu, Binghuai Lin, Yunbo Cao, Qi Liu, Tianyu Liu, and Zhifang Sui. 2023c. Large language models are not fair evaluators. *arXiv preprint arXiv:2305.17926*.

Yue Wang, Hung Le, Akhilesh Deepak Gotmare, Nghi D.Q. Bui, Junnan Li, and Steven C. H. Hoi. 2023d. CodeT5+: Open code large language models for code understanding and generation. *arXiv preprint arXiv:2305.07922*.

Zhiheng Xi, Wenxiang Chen, Xin Guo, Wei He, Yiwen Ding, Boyang Hong, Ming Zhang, Junzhe Wang, Senjie Jin, Enyu Zhou, et al. 2023. The rise and potential of large language model based agents: A survey. *arXiv preprint arXiv:2309.07864*.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R Narasimhan, and Yuan Cao. 2023. ReAct: Synergizing reasoning and acting in language models. In *The Eleventh International Conference on Learning Representations*.

Daoguang Zan, Bei Chen, Yongshun Gong, Junzhi Cao, Fengji Zhang, Bingchao Wu, Bei Guan, Yilong Yin, and Yongji Wang. 2023. Private-library-oriented code generation with large language models. *arXiv preprint arXiv:2307.15370*.

Fengji Zhang, Bei Chen, Yue Zhang, Jin Liu, Daoguang Zan, Yi Mao, Jian-Guang Lou, and Weizhu Chen. 2023. RepoCoder: Repository-level code completion through iterative retrieval and generation. *arXiv preprint arXiv:2303.12570*.

Haozhe Zhao, Zefan Cai, Shuzheng Si, Xiaojian Ma, Kaikai An, Liang Chen, Zixuan Liu, Sheng Wang, Wenjuan Han, and Baobao Chang. 2023a. MMICL: Empowering vision-language model with multi-modal in-context learning. *arXiv preprint arXiv:2309.07915*.

Wayne Xin Zhao, Kun Zhou, Junyi Li, Tianyi Tang, Xiaolei Wang, Yupeng Hou, Yingqian Min, Beichen Zhang, Junjie Zhang, Zican Dong, et al. 2023b. A survey of large language models. *arXiv preprint arXiv:2303.18223*.

Qinkai Zheng, Xiao Xia, Xu Zou, Yuxiao Dong, Shan Wang, Yufei Xue, Zihan Wang, Lei Shen, Andi Wang, Yang Li, Teng Su, Zhilin Yang, and Jie Tang. 2023. CodeGeeX: A pre-trained model for code generation with multilingual evaluations on HumanEval-X. In *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, page 5673–5684, New York, NY, USA.

A Related work

A.1 Code Generation

Code generation has been a subject of longstanding inquiry in the field of NLP. It demonstrates activity in the formulation of methodologies and the establishment of benchmarks (Chen et al., 2021a; Christopoulou et al., 2022; Orlanski et al., 2023; Wang et al., 2023d; Cassano et al., 2022; Tang et al., 2023a,b). While existing code benchmarks target advancing the capability for function-level code generation, ML-BENCH focuses on leveraging code generation to facilitate tool usage. More details are shown at Tab. 4

Function-Level Code Generation The objective of function-level code generation is to produce code snippets meeting user requirements or contribute to code completion (Feng et al., 2020; Li et al., 2022), encompassing the development of foundational models for code generation (Zheng et al., 2023; AI, 2023).

Tool-using. Tool-using code generation empowers the model's ability of tool calling, enabling it to acquire the proficiency to invoke tools when engaging with the user, like ToolLLM (Qin et al., 2023), Gorilla (Patil et al., 2023) and HuggingGPT (Shen et al., 2023).

A.2 Agent

The rapid advancement of usages (Yao et al., 2023; Li et al., 2023; Wang et al., 2023c; Cai et al., 2023) for large language models (Brown et al., 2020; Shao et al., 2022; Zhao et al., 2023a) has propelled the evolution of intelligent agents within the realm of artificial intelligence (Zhao et al., 2023b; Xi et al., 2023). The development of agents based on LLMs encompasses two primary facets: general agents and task-specific agents.

General Agents. General agents refer to agents who are not explicitly assigned a predefined task during construction. This category of agents demonstrates the capability to autonomously undertake a diverse range of complex tasks through mechanisms such as planning, reaction, memorization, reasoning, and the establishment of communication protocols (Yao et al., 2023; Wang et al., 2023b), like AutoGPT (Gravitas, 2023), XAgent (Team, 2023), AgentVerse (Chen et al., 2023b), HOLMES (Chen et al., 2023a) and MetaGPT (Hong et al., 2023).

Task-specific Agents. Task-specific agents pertain to agents meticulously engineered for the explicit purpose of undertaking a specific task. Illustrative instances encompass MindAgent (Gong et al., 2023), designed to augment collaborations between humans and NPCs in games, and Voyager (Wang et al., 2023a), playing Minecraft.

B Details of Selected GitHub Repositries

Tab.5 shows details of selected GitHub repositories.

Tab.6 shows characteristics of selected GitHub repositories.

C Details

Tab.8 shows the detailed length of tokens for each repository.

D Data Construction of ML-BENCH

Tab.9 shows the data construction of ML-BENCH.

E Statistics of ML-BENCH

Tab.9 shows the statistics of ML-BENCH.

F Characteristics for Repositories

Tab.10 shows the characteristics of repositories, the sufficiency of information means whether the README has sufficient information to generate output. The sparsity of code demonstrates the ratio of text and code; dense means README files have more code than text, while text is more than code. The concentration of information illustrates whether the information is centralized; the higher the number is, the more centralized the files are; 0 means some samples have no information to generate output in README files in the repository.

We hope this information can help distinguish different difficulties for different repositories.

Domain	GitHub	Star	URL	#README	Last Updated
Text	BERT	35693	https://github.com/google-research/bert	1	2020.03.11
	Tensor2Tensor	14280	https://github.com/tensorflow/tensor2tensor	9	2023.04.01
	Text Classification	7665	https://github.com/brightmart/text_classification	1	2022.09.21
GNN	GNN	12429	https://github.com/dmlc/dgl	154	2023.11.16
molecular	ESM	2462	https://github.com/facebookresearch/esm	8	2023.06.27
Image-GAN	PyTorch-GAN	14947	https://github.com/eriklindernoren/PyTorch-GAN	1	2021.01.07
Multi-Modality	LAVIS	7300	https://github.com/salesforce/lavis	8	2023.09.25
	IF	7237	https://github.com/deep-floyd/if	1	2023.06.03
	OPEN CLIP	6856	https://github.com/mlfoundations/open_clip	1	2023.11.01
	Stable	31506	https://github.com/Stability-AI/stablediffusion	1	2023.03.25
	Segment-Anything	11976	https://github.com/IDEA-Research/Grounded-Segment-Anything	3	2023.12.11
Music	Muzic	3866	https://github.com/microsoft/muzic	8	2023.12.06
3D	Learning3d	579	https://github.com/vinits5/learning3d	1	2023.10.24
Video	Vid2Vid	8393	https://github.com/NVIDIA/vid2vid	2	2019.07.04
Time-Series	Time-Series-Library	2670	https://github.com/thuml/Time-Series-Library	1	2023.11.10
Attention Usage	External-Attention-pytorch	9949	https://github.com/xmu-xiaoma666/External-Attention-pytorch	1	2023.10.25
Medicine	MedicalZooPytorch	1516	https://github.com/black0017/MedicalZooPytorch	21	2022.02.07

Table 5: Detailed information about the selected GitHub repositories.

GitHub	sufficiency of information	sparsity of information	concentration of information	Last Updated
BERT	Yes	Dense	1	22 Bash Script
Tensor2Tensor	14280	https://github.com/tensorflow/tensor2tensor	9	2023.04.01
Text Classification	7665	https://github.com/brightmart/text_classification	1	2022.09.21
GNN	No	Sparse	3	21 Bash Scirpt
ESM	Yes	Dense	2	15 Bash Scirpt, 2 Python Code
PyTorch-GAN	No	Sparse	3	30 Bash Script
LAVIS	Yes	Dense	1	4 Bash Script, 23 Python Code
IF	Yes	Dense	2	21 Bash Script
OPEN CLIP	Yes	Dense	3	11 Python Code
Stable	31506	https://github.com/Stability-AI/stablediffusion	1	2023.03.25
Segment-Anything	11976	https://github.com/IDEA-Research/Grounded-Segment-Anything	3	20 Bash Script
Muzic	3866	https://github.com/microsoft/muzic	8	17 Bash Script
Learning3d	579	https://github.com/vinits5/learning3d	1	17 Python Code
Vid2Vid	Yes	Dense	3	13 Python Code
Time-Series-Library	No	Sparse	3	14 Python Code
External-Attention-pytorch	Yes	Dense	3	24 Python Code
MedicalZooPytorch	1516	https://github.com/black0017/MedicalZooPytorch	21	2022.02.07
Pytorch-IM	1516	https://github.com/black0017/MedicalZooPytorch	21	5 Bash Script

Table 6: Characteristic information about the selected GitHub repositories.

	Raw README	Oracle Segment	BM25
Train			
Average token length	8009.42	151.32	492.90
Max token length	20701	851	5540
OOD Train			
Average token length	4118.90	186.04	585.80
Max token length	8363	367	1339
Test			
Average token length	7842.47	1186.37	1402.24
Max token length	20655	5420	3192

Table 7: The statistics of README files.

Repos	CodeLlama						GPT 3.5					
	Raw README		BM25 retrieval		Oracle segment		Raw README		BM25 retrieval		Oracle segment	
	Train	Test	Train	Test	Train	Test	Train	Test	Train	Test	Train	Test
vid2vid	4468	4669	408	1615	556	556	3569	3807	338	1240	416	416
If	3972	4202	1390	1065	3023	3023	3042	3320	1119	851	2367	2367
DGL	7401	9668	312	2598	179	138	5924	7194	275	1924	143	110
Py-GAN	10619	10644	532	515	314	314	9067	9092	433	424	268	268
ESM	11955	12705	585	1419	177	173	9218	10101	486	1168	139	136
BERT	13373	14309	401	985	372	375	10558	11323	335	783	287	290
OpenCLIP	8005	8403	415	1517	5420	5420	6340	6813	350	1202	4397	4397
Lavis	4447	4779	471	1735	1984	1984	3377	3699	372	1347	1547	1547
TSL	1891	2008	382	1136	345	345	1460	1598	315	918	276	276
EAP	20701	20655	1155	473	105	118	14591	14908	857	410	69	80
Grounded-SAM	/	9130	/	1557	/	164	/	6741	/	1196	/	113
Py-IM	/	19701	/	8929	/	89	/	14192	/	6061	/	68
muzic	/	2150	/	800	/	83	/	1727	/	625	/	64
Learning3d	/	3952	/	1163	/	50	/	2970	/	896	/	45
SD	4203	/	471	/	234	/	3341	/	393	/	183	/
MedZooPy	2393	/	1101	/	133	/	1868	/	879	/	99	/
TCL	8363	/	304	/	116	/	7066	/	278	/	96	/
Tensor2Tensor	4566	/	476	/	192	/	3673	/	399	/	153	/

Table 8: Detailed length of tokens for each repository.

G Detailed repository scores for models

Fig.5 shows the repositories error analysis results for ML-AGENT and GPT-4.

H An Illustration of Input-Output Pairs

Fig.6 shows an illustration of input-output pairs and its evaluation result.

I BLEU Scores and Cutoff Date

Tab.11 shows BLEU Scores between Oracle Segment and BM25 contents, and Tab.12 shows cutoff date of training data for the models.

J Structure of ML-AGENT

Fig.7 shows the structure of ML-AGENT

K Template

Tab.13 shows template for LLMs, Tab.14 shows template for fine-tuning LLMs, Tab.15 shows template for Retrieval step in ML-AGENT, Tab.16 shows template for Ranking step in ML-AGENT, Tab.17 shows template for Planning step in ML-AGENT, Tab.18 shows template for Generating step in ML-AGENT,

L Examples of input-output of each GitHub Repo

In this section, we present detailed examples of the input and output of each GitHub Repo in Tab.19 to Tab.36. The corresponding repository for each table is shown below:

1. External-Attention: Tab. 19
2. BERT: Tab. 20
3. Deep learning on graphs: Tab. 21
4. Evolutionary scale modeling: Tab. 22
5. Grounded-Segment-Anything: Tab. 23
6. DeepFloyd IF: Tab. 24
7. Language-Vision Intelligence: Tab. 25
8. Deep learning on 3D point clouds data: Tab. 26
9. 3D multi-modal medical image segmentation library: Tab. 27
10. Music understanding and generation: Tab. 28
11. Implementation of OpenAI's CLIP: Tab. 29
12. Generative Adversarial Network varieties: Tab. 30

Task	Number
- GNN	608
- Text	1475
- Molecular	649
- Image-GAN	1189
- Multi-Modality	4732
- Video	75
- Time-series	1478
- Attention Usage	127
- Medical	805
- 3D	264
- Music	704
Average token length per instruction	80.4
Max token length in instruction	216
Instruction edit distance among the same task	258.7
Instruction edit distance across tasks	302.1

Table 9: The statistics of ML-BENCH.

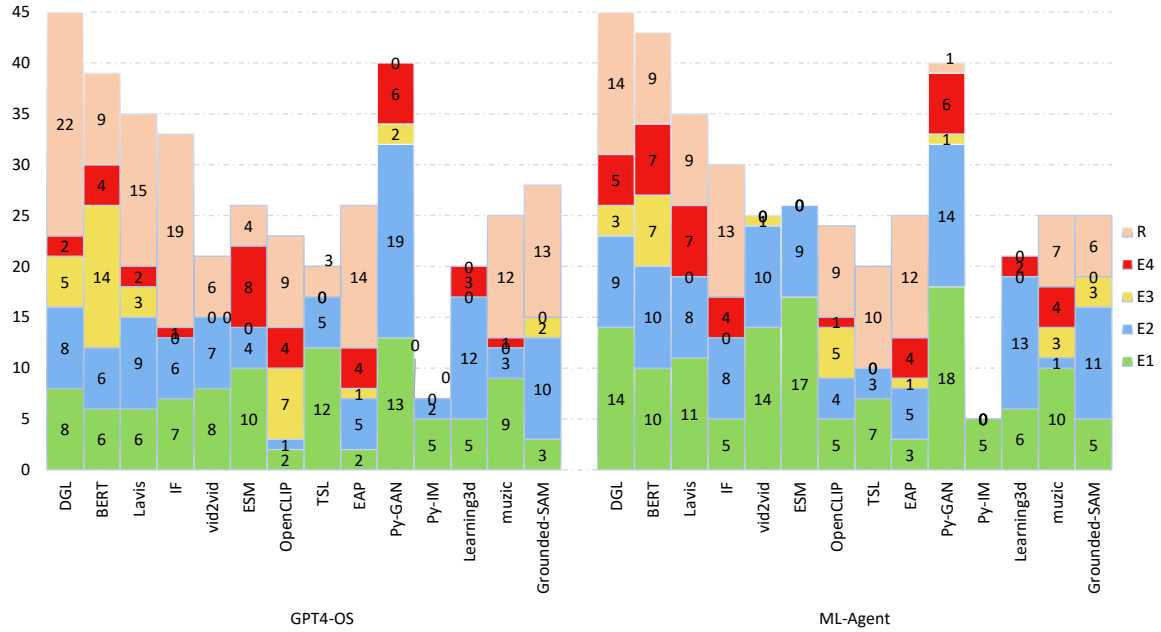

Figure 5: Quantification of errors for five times attempts in different repositories perspective, the result for GPT-4 is under the Oracle Segment setting.

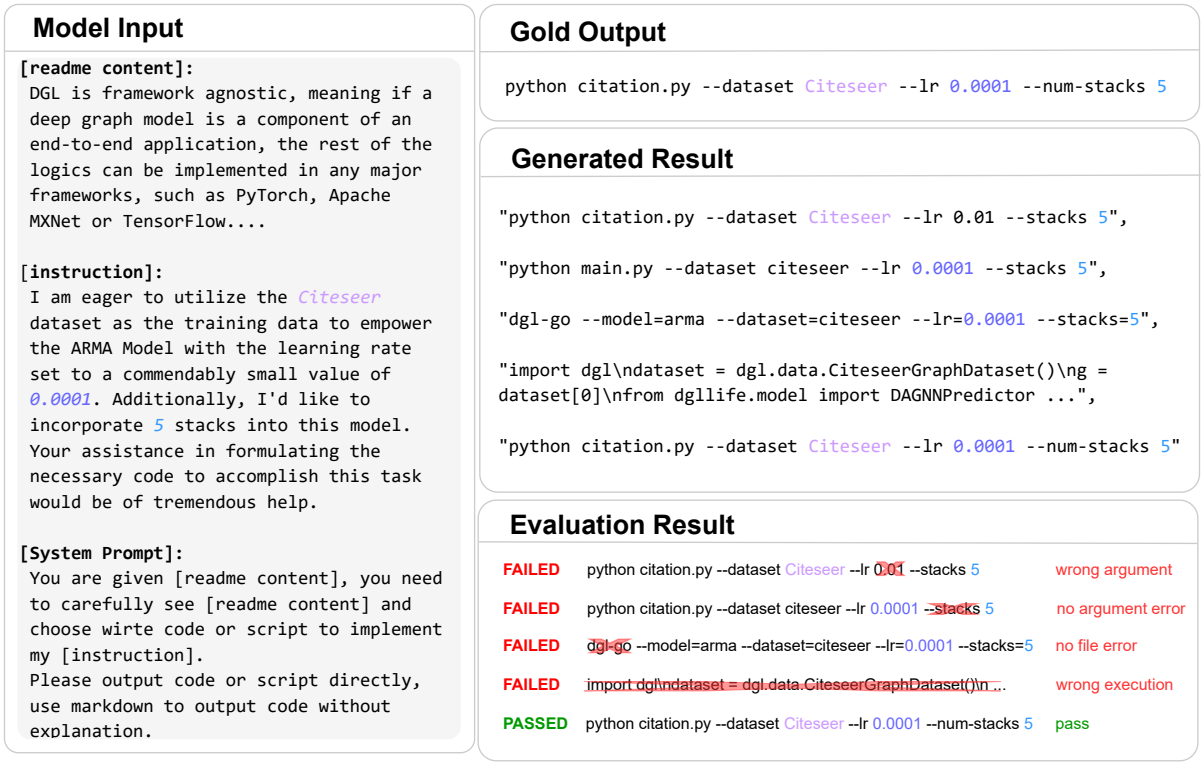

Figure 6: The illustrative set of **input-output pairs, gold output, and execution results**, accompanied by precision metrics: **Pass@1=0, Pass@2=0, and Pass@5=1**. Various colors within the instructions signify different parameters.

	sufficiency of information	sparsity of code	concentration of information
vid2vid	Yes	Dense	3
If	Yes	Dense	2
DGL	No	Sparse	3
Py-GAN	No	Sparse	3
ESM	Yes	Dense	2
BERT	Yes	Dense	1
OpenCLIP	Yes	Dense	2
Lavis	Yes	Dense	3
TSL	No	Sparse	0
EAP	Yes	Dense	3
Py-IM	Yes	Sparse	2
muzic	Yes	Dense	3
Learing3d	No	Sparse	0
Grounded-SAM	No	Dense	2

Table 10: The characteristics of repositories

13. PyTorch Image Models: Tab. 31
14. Stable diffusion: Tab. 32
15. Text classification: Tab. 33
16. Tensor2Tensor: Tab. 34
17. deep time series analysis: Tab. 35

Table 11: The BLEU-1 scores between Oracle Segments and the BM25 Retrieval content.

	ID-train	OOD-train	ML-BENCH
BLEU score	0.0112	0.0087	0.0082

Table 12: Cutoff date of training data for GPTs and Claude 2

	GPT-3.5	GPT-4	Claude 2	CodeLlama
Datetime	2021.09	2021.09	Early 2023	2023.07

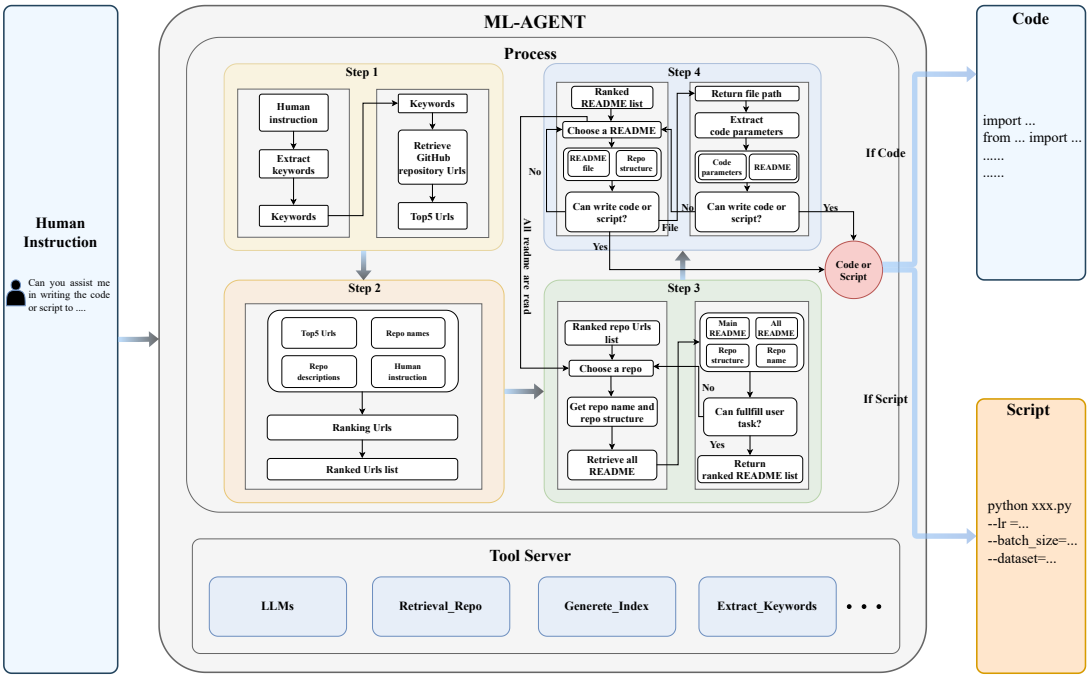

Figure 7: Structure of ML-AGENT. ML-AGENT follows a four-step process upon receiving human instructions: **Step 1**: Extract keywords from the instructions and conduct searches on GitHub to identify the five most relevant repositories. **Step 2**: Sort the retrieved GitHub data. **Step 3**: Evaluate each retrieved repository to determine its capability to fulfil the task. **Step 4**: Write the script or code necessary to implement the task.

System Prompt

You are given [readme], you need to carefully see [readme] and choose wirte code or script to implement my [instruction].
Please output your response in a python markdown code block.

Read Me

[readme]:{Read Me}

Instruction

[instruction]:{Instruction}

Table 13

Read Me

[readme]:{Read Me}

Instruction

[instruction]:{Instruction}

Table 14

Instruction
Please help me extract the keywords in this sentence about the model used and the task I want to do. Sentence :{}

Table 15: The function of this prompt agent is to extract keywords about the model used and the task I want to do from the given sentence.

Instruction
Sort the names and descriptions of the GitHub repositories below based on their relevance to the keywords {keyword}. URLS:{} Make sure your answer is in standard json format, start!
Example

Below is an example:
Sort the names and descriptions of the GitHub repositories below based on their relevance to the keywords {CNN, Time-Series}.
URLS:{
'https://github.com/mrharicot/monodepth',
'https://github.com/saeedkhaki92/CNN-RNN-Yield-Prediction',
'https://github.com/PatientEz/CNN-BiLSTM-Attention-Time-Series-Prediction_Keras',
'https://github.com/hardyqr/CNN-for-Stock-Market-Prediction-PyTorch',
'https://github.com/muskie82/CNN-DSO',
'https://github.com/gognjanovski/StockPredictionCNN',
'https://github.com/kristpapadopoulos/seriesnet',
'https://github.com/Shen-Lab/DeepAffinity',
'https://github.com/matheusbfernandes/stock-market-prediction',
'https://github.com/perseus784/Vehicle_Collision_Prediction_Using_CNN-LSTMs'
}
Make sure your answer is in standard json format, start!

Table 16: Rank github repositories related to keywords. Sort the github repository related to keywords, and the array is arranged.

Instruction
I will provide you with the name of a GitHub repository and the main readme file. Please determine if you can accomplish the following task. If you can, output 'Yes' and sort the provided readme files below based on their helpfulness in completing the task, from most helpful to least helpful. If you believe you cannot complete the task, please output 'No.' repo_name: {repo_name} main_readmefile: {Read Me} task: {Instruction}

Table 17: Use the information given to determine whether the task can be completed. The given information serves as an indicator of whether the task can be completed. If it can be completed, return 'Yes'; if it cannot be completed, return 'No'.eg:No.eg:Yes. Sorted list of readme files based on their helpfulness in completing the task. Remember that the result must be a list!eg:repo_name/README.md,repo_name/examples/README.md,repo_name/scripts/atlas/README.md.

Instruction
I will provide you with a github readme file and directory index, please determine if this information is enough to write the code or script for the task I gave you. Write the code or script directly if you can, give us the path to the py file you want from the repository index file if you need more information.Please make sure that the file path you return is in the repository index file!!!And return NO if you don't think all of this information will do the job. repository_index_file:{} readmefile:{} task:{}

Table 18: Code that uses given information to determine whether or not it can accomplish a given task.

Read Me:

As a supplement to the project, an object detection codebase YOLO. Air has recently been newly opened, which integrates various attention mechanisms in the object detection algorithm. The code is simple and easy to read. Welcome to play and star!
For beginners (like me): Recently, I found a problem when reading the paper. Sometimes the core idea of the paper is very simple, and the core code may be just a dozen lines. However, when I open the source code of the author's release, I find that the proposed module is embedded in the task framework such as classification, detection, and segmentation, resulting in redundant code. For me who is not familiar with the specific task framework, it is difficult to find the core code, resulting in some difficulties in understanding the paper and network ideas.
For advanced (like you): If the basic units conv, FC, and RNN are regarded as small Lego blocks, and the structures transformer and RESNET are regarded as LEGO castles that have been built. The modules provided by this project are LEGO components with complete semantic information. Let scientific researchers avoid repeatedly building wheels, just think about how to use these "LEGO components" to build more colorful works.
For proficient (maybe like you): Limited capacity, do not like light spraying!!!
For All: This project aims to realize a code base that can make beginners of deep learning understand and serve scientific research and industrial communities. As fightingcv WeChat official account. The purpose of this project is to achieve Let there be no hard-to-read papers in the world. (at the same time, we also welcome all scientific researchers to sort out the core code of their work into this project, promote the development of the scientific research community, and indicate the author of the code in readme)
...

Golden Code Segment:

..
from model.attention.ViP import WeightedPermuteMLP
import torch
from torch import nn
from torch.nn import functional as F
...

Instruction:

I'm planning to utilize the fighting-cv model to complete the attention layers for ViP Attention Usage. Could you provide me with some guidance on accomplishing this task?

Instruction:

package_1: ViP
sub_package: WeightedPermuteMLP
package_2: torch

Ground Truth Output:

```
from model.attention.ViP import WeightedPermuteMLP
import torch
from torch import nn
from torch.nn import functional as F
input=torch.randn(64,8,8,512)
seg_dim=8
vip=WeightedPermuteMLP(512,seg_dim)
out=vip(input)
print(out.shape)
```

Table 19: Example of input-output for **External-Attention-pytorch** GitHub on **attention layer** task on **Attention Usage** domain. The ReadMe URL is https://github.com/xmu-xiaoma666/External-Attention-pytorch/blob/master/README_EN.md. The GitHub URL is https://github.com/xmu-xiaoma666/External-Attention-pytorch.

Read Me:

BERT

New March 11th, 2020: Smaller BERT Models

This is a release of 24 smaller BERT models (English only, uncased, trained with WordPiece masking) referenced in Well-Read Students Learn Better: On the Importance of Pre-training Compact Models.

...

Golden Code Segment:

..

This demo code only pre-trains for a smallnumber of steps (20), but in practice you will probably want to set ‘num_train_steps‘ to 10000 steps or more. The ‘max_seq_length‘ and ‘max_predictions_per_seq‘ parameters passed to ‘run_pretraining.py‘ must be the same as ‘create_pretraining_data.py‘.

python run_pretraining.py
--input_file=/tmp/tf_examples.tfrecord
--output_dir=/tmp/pretraining_output
--do_train=True
--do_eval=True
--bert_config_file=$BERT_BASE_DIR/bert_config.json
--init_checkpoint=$BERT_BASE_DIR/bert_model.ckpt

...

Instruction:

Behold, a formidable quest awaits - the pre-training of the unparalleled uncased_L-24_H-1024_A-16 model. Our path to victory lies in configuring the maximum sequence length to a mighty 512, with a pledge to uphold the limit of 30 predictions per sequence. Battling through the treacherous terrain of 10000 steps, we shall march forward, with a stalwart battalion of 32 batch size by our side. But fear not, for we shall brace ourselves with 10000 warmup steps, as we navigate the perilous sea of learning, with a steadfast learning rate of 0.0001. I humbly beseech your assistance, noble comrade, to conjure the code necessary to conquer this heroic endeavor.

Arguments Requirements:

model: ./model/uncased_L-12_H-768_A-16
train_batch_size: 32
max_seq_length: 512
num_train_steps: 10000
num_warmup_steps: 1000
learning_rate: 0.0001

Ground Truth Output:

python run_pretraining.py
--input_file=/tmp/tf_examples.tfrecord
--output_dir=/tmp/pretraining_output
--do_train=True
--do_eval=True
--bert_config_file=./model/uncased_L-24_H-1024_A-16/bert_config.json
--init_checkpoint=./model/uncased_L-24_H-1024_A-16/bert_model.ckpt
--train_batch_size=32
--max_seq_length=512

...

Table 20: Example of input-output for **bert** GitHub on **pre-training** task on **BERT** domain. The ReadMe URL is https://github.com/google-research/bert/blob/master/README.md. The GitHub URL is https://github.com/google-research/bert.

Read Me:

1. DGL Implementation of CorrectAndSmooth

This DGL example implements the GNN model proposed in the paper Combining Label Propagation and Simple Models Out-performs Graph Neural Networks. For the original implementation, see here. Contributor: xnuohz

2. Requirements

The codebase is implemented in Python 3.7. For version requirement of packages, see below.
dgl 0.6.0.post1
torch 1.7.0
ogb 1.3.0
...

Golden Code Segment:

..

3.1 ogbn-arxiv

Plain MLP + C&S

python main.py
–dropout 0.5

python main.py
--pretrain
--correction-adj DA
--smoothing-adj AD
--autoscale
...

Instruction:

...
Together, we shall embark on a noble mission to train the illustrious CorrectAndSmooth Model, fortified with a sublime dropout rate of 0.7. Our arduous journey spans 700 epochs, each pulsating with the promise of enlightenment. Alas, I beseech your sage guidance in the ethereal realm of code crafting, to manifest this grand undertaking.

Arguments Requirements:

dataset: ogbn-arxiv
model: mlp
dropout: 0.7
epochs: 700

Ground Truth Output:

python main.py
--dataset ogbn-arxiv
--model mlp
--dropout 0.7
--epochs 700

Table 21: Example of input-output for **DGL** GitHub on **DGL Implementation of CorrectAndSmooth** task on **GNN** domain. The ReadMe URL is https://github.com/dmlc/dgl/blob/master/examples/pytorch/correct_and_smooth/README.md. The GitHub URL is https://github.com/dmlc/dgl.

Read Me:

Evolutionary Scale Modeling
atlas

Update April 2023: Code for the two simultaneous preprints on protein design is now released! Code for "Language models generalize beyond natural proteins" is under examples/lm-design/. Code for "A high-level programming language for generative protein design" is under examples/protein-programming-language

This repository contains code and pre-trained weights for Transformer protein language models from the Meta Fundamental AI Research Protein Team (FAIR), including our state-of-the-art ESM and ESMFold, as well as MSA Transformer, ESM-1v for predicting variant effects and ESM-IF1 for inverse folding.
...

Golden Code Segment:

..
The following commands allow the extraction of the final-layer embedding for a FASTA file from the ESM-2 model:
esm-extract esm2_t33_650M_UR50D examples/data/some_proteins.fasta
examples/data/some_proteins_emb_esm2
  --repr_layers 0 32 33
  --include

python scripts/extract.py esm2_t33_650M_UR50D examples/data/some_proteins.fasta
examples/data/some_proteins_emb_esm2
  --repr_layers 0 32 33
  --include mean per_tok
  --A cuda device is optional and will be auto-detected.
...

Instruction:

...
Can you assist me in writing the code to extract the 24-layer embedding for a FASTA file named rna.fasta using the esm1v_t33_650M_UR90S_5 model and save the output?

Arguments Requirements:

model: esm1v_t33_650M_UR90S_5
data: rna.fasta
layer_number: 24
layer_name: repr_layers

Ground Truth Output:

python scripts/extract.py esm1v_t33_650M_UR90S_5 rna.fasta output.embeddings
  --repr_layers 24
  --include mean per_tok

Table 22: Example of input-output for **ESM** GitHub on **Extract ESMFold Structure Prediction Model's Embedding** task on **molecular** domain. The ReadMe URL is https://github.com/facebookresearch/esm/blob/master/README.md. The GitHub URL is https://github.com/facebookresearch/esm.

Read Me:

Official PyTorch implementation of Grounding DINO), a stronger open-set object detector. Code is available now!

Highlight

- Open-Set Detection. Detect everything with language!
- High Performancce. COCO zero-shot 52.5 AP (training without COCO data!). COCO fine-tune 63.0 AP.
- Flexible. Collaboration with Stable Diffusion for Image Editing.

...

Golden Code Segment:

..

Demo

python demo/inference_on_a_image.py
  -c /path/to/config
  -p /path/to/checkpoint
  -i .asset/cats.png
  -o outputs/0
  -t cat ear. [--cpu-only] # open it for cpu mode

See the demo/inference_on_a_image.py for more details.

...

Instruction:

...

I am interested in utilizing the grounding dino demo for a specific task. The input image path is ground_segment/GD_new.json, and I would like the output to be saved in the directory output/cat2002. Additionally, I would like the text condition to be set to right ear of cat. Could you kindly assist me in writing the script to achieve this?

Arguments Requirements:

i: .asset/cat.jpg
o: output/cat2002
t: right ear of cat

Ground Truth Output:

python demo/inference_on_a_image.py
  -c model/GroundingDINO_SwinT_OGC.py
  -p model/groundingdino_swint_ogc.pth
  -i .asset/cat.jpg
  -o output/cat2002
  -t right ear of cat

Table 23: Example of input-output for **Grounded-Segment-Anything** GitHub on **Grounding DINO demo** task on **Segment** domain. The ReadMe URL is https://github.com/IDEA-Research/Grounded-Segment-Anything/blob/main/GroundingDINO/README.md. The GitHub URL is https://github.com/IDEA-Research/Grounded-Segment-Anything.

Read Me:

We introduce DeepFloyd IF, a novel state-of-the-art open-source text-to-image model with a high degree of photorealism and language understanding. DeepFloyd IF is a modular composed of a frozen text encoder and three cascaded pixel diffusion modules: a base model that generates 64x64 px image based on text prompt and two super-resolution models, each designed to generate images of increasing resolution: 256x256 px and 1024x1024 px. All stages of the model utilize a frozen text encoder based on the T5 transformer to extract text embeddings, which are then fed into a UNet architecture enhanced with cross-attention and attention pooling. The result is a highly efficient model that outperforms current state-of-the-art models, achieving a zero-shot FID score of 6.66 on the COCO dataset. Our work underscores the potential of larger UNet architectures in the first stage of cascaded diffusion models and depicts a promising future for text-to-image synthesis.

...

Golden Code Segment:

..

II. Zero-shot Image-to-Image Translation

In Style Transfer mode, the output of your prompt comes out at the style of the support_pil_img

from deepfloyd_if.pipelines import style_transfer

result = style_transfer(
t5 = t5, if_I = if_I, if_II = if_II,
...

Instruction:

...
Time to create a visual masterpiece! I am excited to recreate 'image'.jpg into a fascinating rendition of the ink wash style. I am planning to utilize the capabilities of the IF-I-XL-v1.0 model for this endeavor. Can you assist me in crafting the necessary code?

Arguments Requirements:

model: IF-I-XL-v1.0
argument1: image.jpg
argument2: wash

Ground Truth Output:

from deepfloyd_if.modules import IFStageI, IFStageII, StableStageIII
from deepfloyd_if.modules.t5 import T5Embedder
device = 'cuda:1'
if_I = IFStageI('IF-I-XL-v1.0', device=device)
if_II = IFStageII('IF-II-L-v1.0', device=device)
if_III = StableStageIII('stable-diffusion-x4-upscaler', device=device)
t5 = T5Embedder(device='cpu')

style_prompt = a captivating ink wash style
image_path = image.jpg
...

Table 24: Example of input-output for **IF** GitHub on **Zero-shot Image-to-Image Translation** task on **Image and Text** domain. The ReadMe URL is https://github.com/deep-floyd/IF/blob/develop/README.md. The GitHub URL is https://github.com/deep-floyd/if.

Read Me:

Lavis

Lavis - A Library for Language-Vision Intelligence

What's New:

A simple, yet effective, cross-modality framework built atop frozen LLMs that allows the integration of various modalities (image, video, audio, 3D) without extensive modality-specific customization.

Technical Report and Citing LAVIS:

...

Golden Code Segment:

...

how to use models in LAVIS to perform inference on example data. We first load a sample image from local.

import torch

from PIL import Image

setup device to use

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

load sample image

raw_image = Image.open(merlion.png).convert(RGB)

This example image shows Merlion park (source), a landmark in Singapore.

Image Captioning

In this example, we use the BLIP model to generate a caption for the image.

...

Instruction:

There is a task in front of me currently, which is the Image Captioning task. I ask for your kind help in moving forward with this task.

Arguments Requirements:

image: image.jpg

Ground Truth Output:

```
import torch
from PIL import Image
from lavis.models import load_model_and_preprocess
device = torch.device('cuda:1' if torch.cuda.is_available() else 'cpu')
model, vis_processors, _ = load_model_and_preprocess(name='blip_caption',
model_type='base_coco', is_eval=True, device=device)
raw_image = Image.open('image.jpg').convert('RGB')
preprocessed_image = vis_processors['eval'](raw_image).unsqueeze(0).to(device)
output = model.generate({'image': preprocessed_image})
caption = output['captions'][0]['text']
print(caption)
```

Table 25: Example of input-output for **Lavis** GitHub on **Image Captioning** task on **Multimodal Image and Text** domain. The ReadMe URL is https://github.com/salesforce/LAVIS/blob/main/README.md. The GitHub URL is https://github.com/salesforce/lavis.

Read Me:

Learning3D: A Modern Library for Deep Learning on 3D Point Clouds Data.
Learning3D is an open-source library that supports the development of deep learning algorithms that deal with 3D data. The Learning3D exposes a set of state of art deep neural networks in python. A modular code has been provided for further development. We welcome contributions from the open-source community.
Available Computer Vision Algorithms in Learning3D
...

Golden Code Segment:

...
examples/test_dcp.py Learning3D is an open-source library that supports the development of deep learning algorithms that deal with 3D data. The Learning3D exposes a set of state of art deep neural networks in python
python test_dcp.py
--num_points 128
--j 12
--symfn max
...

Instruction:

I am interested in conducting a test using the dcp model. Specifically, I would like to set the parameters as follows: the test mode should be selected, the model should be set to dcp, the number of points should be 512, the number of data loading workers should be -j 8, and the symmetric function should be set to –symfn max. Could you please assist me in writing the code or script necessary to carry out this test?

Arguments Requirements:

number of points: 512
number of data loading workers: 8
symmetric function: max

Ground Truth Output:

python test_dcp.py
--num_points 512
--j 8
--symfn max

Table 26: Example of input-output for **Learning3D** GitHub on **Test dcp model** task on **3D** domain. The ReadMe URL is https://github.com/vinits5/learning3d/blob/master/README.md. The GitHub URL is https://github.com/vinits5/learning3d.

Read Me:
MusicBERT Basics All models accept two parameters: a) the input the channels (in_channels), and b) the segmentation classes (classes) and produce un-normalized outputs All losses accept as input the prediction in 5D shape of [batch,classes,dim_1,dim_2,dim_3] and the target in 4D target shape of [batch, dim_1, dim_2, dim_3]. It is converted to one-hot inside the loss function for consistency reasons. Furthermore the normalization of the predictions is handled here. Dice-based losses return the scalar loss for backward(), and the prediction per channels in numpy to track training progress. ...
Golden Code Segment:
Usage How to train your model For Iseg-2017 : python ./examples/train_iseg2017_new.py --args For MR brains 2018 (4 classes) python ./examples/train_mrbrains_4_classes.py --args For MR brains 2018 (8 classes) python ./examples/train_mrbrains_9_classes.py --args For MICCAI 2019 Gleason Challenge python ./examples/test_miccai_2019.py --args The arguments that you can modify are extensively listed in the manual.
Instruction:
I'm seeking assistance in writing a piece of code that can successfully train a model for the 'Iseg 2017 Task'. The model in question is 'RESNET3DVAE' and I require the learning rate to be set to '1e-3'. It is also crucial that the training samples are set to '10'. Lastly, use 'sgd' as the optimizer. Could you kindly help out in creating this algorithm?
Arguments Requirements:
lr: 1e-3 samples_train: 10 model: RESNET3DVAE soptimizer: sg
Ground Truth Output:
python ./examples/train_iseg2017_new.py --lr 1e-3 --samples_train 10 --model RESNET3DVAE --opt sgd

Table 27: Example of input-output for **MedicalZoo** GitHub on **Iseg-2017** task on **Medical** domain. The ReadMe URL is https://github.com/black0017/MedicalZooPytorch/blob/master/manual/README.md. The GitHub URL is https://github.com/black0017/MedicalZooPytorch.

Read Me:
MusicBERT MusicBERT: Symbolic Music Understanding with Large-Scale Pre-Training, by Mingliang Zeng, Xu Tan, Rui Wang, Zeqian Ju, Tao Qin, Tie-Yan Liu, ACL 2021, is a large-scale pre-trained model for symbolic music understanding. It has several mechanisms including OctupleMIDI encoding and bar-level masking strategy that are specifically designed for symbolic music data, and achieves state-of-the-art accuracy on several music understanding tasks, including melody completion, accompaniment suggestion, genre classification, and style classification. Projects using MusicBERT: midiformers: a customized MIDI music remixing tool with easy interface for users. **1. Preparing datasets** **1.1 Pre-training datasets** Prepare tar -xzvf lmd_full.tar.gz zip -r lmd_full.zip lmd_full Run the dataset processing script. (‘preprocess.py‘) python -u preprocess.py The script should prompt you to input the path of the midi zip and the path for OctupleMIDI output. ...
Golden Code Segment:
Pre-training bash train_mask.sh lmd_full small Download our pre-trained checkpoints here: small and base, and save in the checkpoints folder. (a newer version of fairseq is needed for using provided checkpoints: see issue-37 or issue-45)
Instruction:
I am interested in conducting a test using the dcp model. Specifically, I would like to set the parameters as follows: the test mode should be selected, the model should be set to dcp, the number of points should be 512, the number of data loading workers should be -j 8, and the symmetric function should be set to –symfn max. Could you please assist me in writing the code or script necessary to carry out this test?
Arguments Requirements:
bash: train_mask.sh dataset: lmd_full checkpoint: small
Ground Truth Output:
bash train_mask.sh lmd_full small

Table 28: Example of input-output for **Muzic** GitHub on **Pre-training model** task on **Music** domain. The ReadMe URL is https://github.com/microsoft/muzic/blob/main/musicbert/README.md. The GitHub URL is https://github.com/microsoft/muzic.

Read Me:

OpenCLIP

Welcome to an open source implementation of OpenAI's CLIP(Contrastive Language-Image Pre-training).

Using this codebase, we have trained several models on a variety of data sources and compute budgets, ranging from small-scale experiments to larger runs including models trained on datasets such as LAION-400M, LAION-2B and DataComp-1B.

Many of our models and their scaling properties are studied in detail in the paper reproducible scaling laws for contrastive language-image learning.

Some of our best models and their zero-shot ImageNet-1k accuracy are shown below, along with the ViT-L model trained by OpenAI. We provide more details about our full collection of pretrained models here, and zero-shot results for 38 datasets here.

midiformers: a customized MIDI music remixing tool with easy interface for users.

...

Golden Code Segment:

Pre-training bash train_mask.sh lmd_full small Download our pre-trained checkpoints here: small and base, and save in the checkpoints folder. (a newer version of fairseq is needed for using provided checkpoints: see issue-37 or issue-45)

Instruction:

I'm hoping you could assist me with a task. I need to sort a collection of images located at the directory '. /image'. For this purpose, I'll be using the 'ViT-B-32' model along with the pretraining model 'commonpool_s_laion_s13m_b4k'. Would you be able to provide instructions or steps on how to accomplish this task?

Arguments Requirements:

model: ViT-B-32
pretraining_model: commonpool_s_laion_s13m_b4k
path: ./image

Ground Truth Output:

```
import os
import numpy as np
import torch
import open_clip
open_clip.list_pretrained()
model, _, preprocess =
open_clip.create_model_and_transforms('ViT-B-32', pretrained='commonpool_s_laion_s13m_b4k')
model.eval()
context_length = model.context_length
vocab_size = model.vocab_size
...
```

Table 29: Example of input-output for **OpenCLIP** GitHub on **Image classification** task on **Image and Text** domain. The ReadMe URL is https://github.com/mlfoundations/open_clip/blob/main/README.md. The GitHub URL is https://github.com/mlfoundations/open_clip.

Read Me:

MusicBERT

Collection of PyTorch implementations of Generative Adversarial Network varieties presented in research papers. Model architectures will not always mirror the ones proposed in the papers, but I have chosen to focus on getting the core ideas covered instead of getting every layer configuration right. Contributions and suggestions of GANs to implement are very welcomed.

Table of Contents:

Installation

Implementations

Auxiliary Classifier GAN

Adversarial Autoencoder

...

Golden Code Segment:

...

Energy-Based GAN

Among them, we show one instantiation of EBGAN framework as using an auto-encoder architecture, with the energy being the reconstruction error, in place of the discriminator. We show that this form of EBGAN exhibits more stable behavior than regular GANs during training. We also show that a single-scale architecture can be trained to generate high-resolution images.

Run Example

$ cd implementations/ebgan/

$ python3 ebgan.py

Instruction:

I have a task to work with the Energy-Based GAN model. The learning rate for this task needs to be set at 0.0001, the number of training epochs should be defined as 100, and the batch size should be fixed at 16. Furthermore, I want the image size to be set at 128. Can you please assist me in framing the script to facilitate this?

Arguments Requirements:

lr: 0.0001

n_epochs: 100

batch_size: 16

img_size: 128

model: ebgan

Ground Truth Output:

python3 ebgan.py

--lr 0.0001

--n_epochs 100

--batch_size 16

--mg_size 128

Table 30: Example of input-output for **pyGAN** GitHub on **Energy-Based GAN** task on **images-many-GANs** domain. The ReadMe URL is https://github.com/eriklindernoren/PyTorch-GAN/blob/master/README.md. The GitHub URL is https://github.com/eriklindernoren/PyTorch-GAN.

Read Me:

PyTorch Image Models

...

What's new

...

Introduction

PyTorch Image Models (timm) is a collection of image models, layers, utilities, optimizers, schedulers, data-loaders / augmentations, and reference training / validation scripts that aim to pull together a wide variety of SOTA models with ability to reproduce ImageNet training results.

...

Golden Code Segment:

..

Existing method of changing patch_size (resize pretrained patch_embed weights once) on creation still works.

Example validation cmd

python validate.py /imagenet
--model vit_base_patch16_224
--amp
--amp-dtype bfloat16
--img-size 255
--crop-pct 1.0
--model-kwargs dynamic_img_size=True dyamic_img_pad=True

...

Instruction:

I am interested in performing the task of resizing the image or window. For this purpose, I would like to utilize the model vit_base_patch16_224. Additionally, it would be helpful to set the amp-dtype to bfloat16. Moreover, I would like to specify the image size as 255 and the crop percentage as 1.0. To ensure flexibility, I would like to enable dynamic image size and dynamic image padding. Could you kindly assist me in creating the code or script to accomplish this objective?

Arguments Requirements:

model: vit_base_patch16_224
amp-dtype: bfloat16
img-size: 255
crop-pct: 1.0
dynamic_img_size: True
dyamic_img_pad: True

Ground Truth Output:

python validate.py /imagenet
--model vit_base_patch16_224
--amp
--amp-dtype bfloat16
--img-size 255
--crop-pct 1.0
--model-kwargs dynamic_img_size=True

Table 31: Example of input-output for **PyIM** GitHub on **PyIM Implementation of Resize The Image/Window** task on **Image** domain. The ReadMe URL is https://github.com/huggingface/pytorch-image-models/blob/main/README.md. The GitHub URL is https://github.com/huggingface/pytorch-image-models.

Read Me:

Stable Diffusion Version 2

This repository contains Stable Diffusion models trained from scratch and will be continuously updated with new checkpoints. The following list provides an overview of all currently available models. More coming soon.

...

Requirements

You can update an existing latent diffusion environment by running.

...

Golden Code Segment:

...

We provide the configs for the SD2-v (768px) and SD2-base (512px) model. First, download the weights for SD2.1-v and SD2.1-base. To sample from the SD2.1-v model, run the following:

python scripts/txt2img.py
--prompt "a professional photograph of an astronaut riding a horse"
--ckpt <path/to/768model.ckpt\>
--config configs/stable-diffusion/v2-inference-v.yaml
--H 768
--W 768

or try out the Web Demo: Hugging Face Spaces.

...

Instruction:

...

For the task of generating an image from text, I need your assistance in writing the code. We'll be using the scripts/txt2img.py script along with the SD2.1-v model. Ensure that the model checkpoint file is located at ..... As we want to generate a high-quality image, set the number of sampling steps to 20. The prompt to generate the image is "a professional photograph of an astronaut riding a horse" and we only need one iteration of the generation process. Can you help me write the code to accomplish this task?

Arguments Requirements:

repeat: 1
config: "configs/stable-diffusion/v2-inference-v.yaml"
ckpt: "ckpt/SD2_1_v_model.ckpt"
prompt: "a professional photograph of an astronaut riding a horse"
precision: full
steps: 20
seed: 2048

Ground Truth Output:

python scripts/txt2img.py
--prompt "a professional photograph of an astronaut riding a horse"
--ckpt ckpt/SD2_1_v_model.ckpt
--config configs/stable-diffusion/v2-inference-v.yaml
--H 768
--W 768
--seed 2048
--precision full
--steps 20
--repeat 1

Table 32: Example of input-output for **SD** GitHub on **SD Implementation of Text-to-Image** task on **Stable Diffusion** domain. The ReadMe URL is https://github.com/Stability-AI/stablediffusion/blob/main/README.md. The GitHub URL is https://github.com/Stability-AI/stablediffusion.

Read Me:

Text Classification

The purpose of this repository is to explore text classification methods in NLP with deep learning.

...

Usage:

1.model is in xxx_model.py

2.run python xxx_train.py to train the model

...

Golden Code Segment:

it learn represenation of each word in the sentence or document with left side context and right side context:

representation current word=[left_side_context_vector,current_word_embedding,right_side_context_vecotor].

for left side context, it use a recurrent structure, a no-linearity transfrom of previous word and left side previous context; similarly to right side context.check: p71_TextRCNN_model.py

Instruction:

I am looking to utilize the TextRCNN model for a particular task. In the course of executing this task, I would like to fix the learning rate at 0.00001, the number of training epochs at 300, and set my batch size to 16. Are you in a position to assist me in creating the appropriate coding syntax for this purpose?

Arguments Requirements:

model: TextRCNN

learning_rate: 0.00001

num_epochs: 300

batch_size: 16

Ground Truth Output:

python3 a04_TextRCNN/p71_TextRCNN_train.py

--num_epochs 300

--batch_size 16

--lr 0.00001

Table 33: Example of input-output for **TC** GitHub on **TC Implementation of TextRCNN** task on **texts** domain. The ReadMe URL is https://github.com/brightmart/text_classification/blob/master/README.md. The GitHub URL is https://github.com/brightmart/text_classification.

Read Me:

Tensor2Tensor

Tensor2Tensor, or T2T for short, is a library of deep learning models and datasets designed to make deep learning more accessible and accelerate ML research.

...

Quick Start

This iPython notebook explains T2T and runs in your browser using a free VM from Google, no installation needed. Alternatively, here is a one-command version that installs T2T, downloads MNIST, trains a model and evaluates it:

...

Golden Code Segment:

...

Test the quick-start on a Workspace's Terminal with this command

t2t-trainer –generate_data
  --data_dir=./t2t_data
  --output_dir=./t2t_train/mnist
  --problem=image_mnist
  --model=shake_shake
  --hparams_set=shake_shake_quick
  --train_steps=1000
  --eval_steps=100

Note: Ensure compliance with the FloydHub Terms of Service.

...

Instruction:

I'd like assistance in trying to make sense of mathematical language. Any help you can give would be highly appreciated, in comprehending the intricate nature of mathematical terminologies and symbols.

Ground Truth Output:

t2t-trainer –generate_data
  --data_dir= /t2t_data
  --output_dir= /t2t_train/mathematical_language
  --problem=algorithmic_math_two_variables
  --model=transformer
  --hparams_set=transformer_tiny
  --train_steps=1000
  --eval_steps=100

Table 34: Example of input-output for **tensor2** GitHub on **tensor2 Implementation of Mathematical Language Understanding** task on **Text** domain. The ReadMe URL is https://github.com/tensorflow/tensor2tensor/blob/master/README.md. The GitHub URL is https://github.com/tensorflow/tensor2tensor.

Read Me:

Time Series Library (TSlib)

TSlib is an open-source library for deep learning researchers, especially for deep time series analysis. We provide a neat code base to evaluate advanced deep time series models or develop your model, which covers five mainstream tasks: long- and short-term forecasting, imputation, anomaly detection, and classification.

...

Usage

1.Install Python 3.8. For convenience, execute the following command.

pip install -r requirements.txt

...

Golden Code Segment:

...

Train and evaluate model. We provide the experiment scripts for all benchmarks under the folder ./scripts/. You can reproduce the experiment results as the following examples:

long-term forecast

bash ./scripts/long_term_forecast/ETT_script/TimesNet_ETTh1.sh

short-term forecast

bash ./scripts/short_term_forecast/TimesNet_M4.sh

...

Instruction:

I'm aiming to conduct a performance evaluation of my FEDformer model which is designated for a classification task. I would like to utilize subset: FaceDetection of the UEA benchmark dataset for this evaluation. The patience parameter needs to be set to 10, with the train epochs set to 16, and batch size adjusted to 100. Could you kindly provide me the code necessary to accomplish this?

Arguments Requirements:

task_name: classification
model: FEDformer
data: UEA
model_id: FaceDetection
patience: 10
batch_size: 16
train_epochs: 100

Ground Truth Output:

python -u run.py
--task_name classification
--model FEDformer
--data UEA
--e_layers 3
--batch_size 16
--d_model 128
--d_ff 256
--top_k 3
--train_epochs 100
--patience 10
...

Table 35: Example of input-output for **tsl** GitHub on **tsl Implementation of classification** task on **Time-series** domain. The ReadMe URL is https://github.com/thuml/Time-Series-Library/blob/main/README.md. The GitHub URL is https://github.com/thuml/Time-Series-Library.

Read Me:

vid2vid

...

Pytorch implementation for high-resolution (e.g., 2048x1024) photorealistic video-to-video translation. It can be used for turning semantic label maps into photo-realistic videos, synthesizing people talking from edge maps, or generating human motions from poses. The core of video-to-video translation is image-to-image translation. Some of our work in that space can be found in pix2pixHD and SPADE.

...

Getting Started

...

Golden Code Segment:

...

To test the model (bash ./scripts/face/test_512.sh): #!./scripts/face/test_512.sh

python test.py

--name edge2face_512

--dataroot datasets/face/

--dataset_mode face

--input_nc 15

--loadSize 512

--use_single_G

The test results will be saved in: ./results/edge2face_512/test_latest/.

Instruction:

We have the task of training the edge2face model on a face dataset. Let's set the resolution size this time to 256. Can you provide assistance in drafting the applicable coding structure?

Arguments Requirements:

model: edge2face

dataset: face

resolusion size: 256

argument: g1

Ground Truth Output:

python train.py

--name edge2face_256_g1

--dataroot datasets/face/

--dataset_mode face

--input_nc 15

--loadSize 256

--ngf 64

--max_frames_per_gpu 6

--n_frames_total 12

--niter 20

--niter_decay 20

Table 36: Example of input-output for **vid2vid** GitHub on **vid2vid Implementation of Edge to Face Training** task on **Videos** domain. The ReadMe URL is https://github.com/NVIDIA/vid2vid/blob/master/README.md. The GitHub URL is https://github.com/NVIDIA/vid2vid.