EvoCodeBench: An Evolving Code Generation Benchmark with Domain-Specific Evaluations

Jia Li, Ge Li, Xuanming Zhang, Yunfei Zhao, Yihong Dong, Zhi Jin, Binhua Li, Fei Huang, Yongbin Li

Peking University Alibaba Group

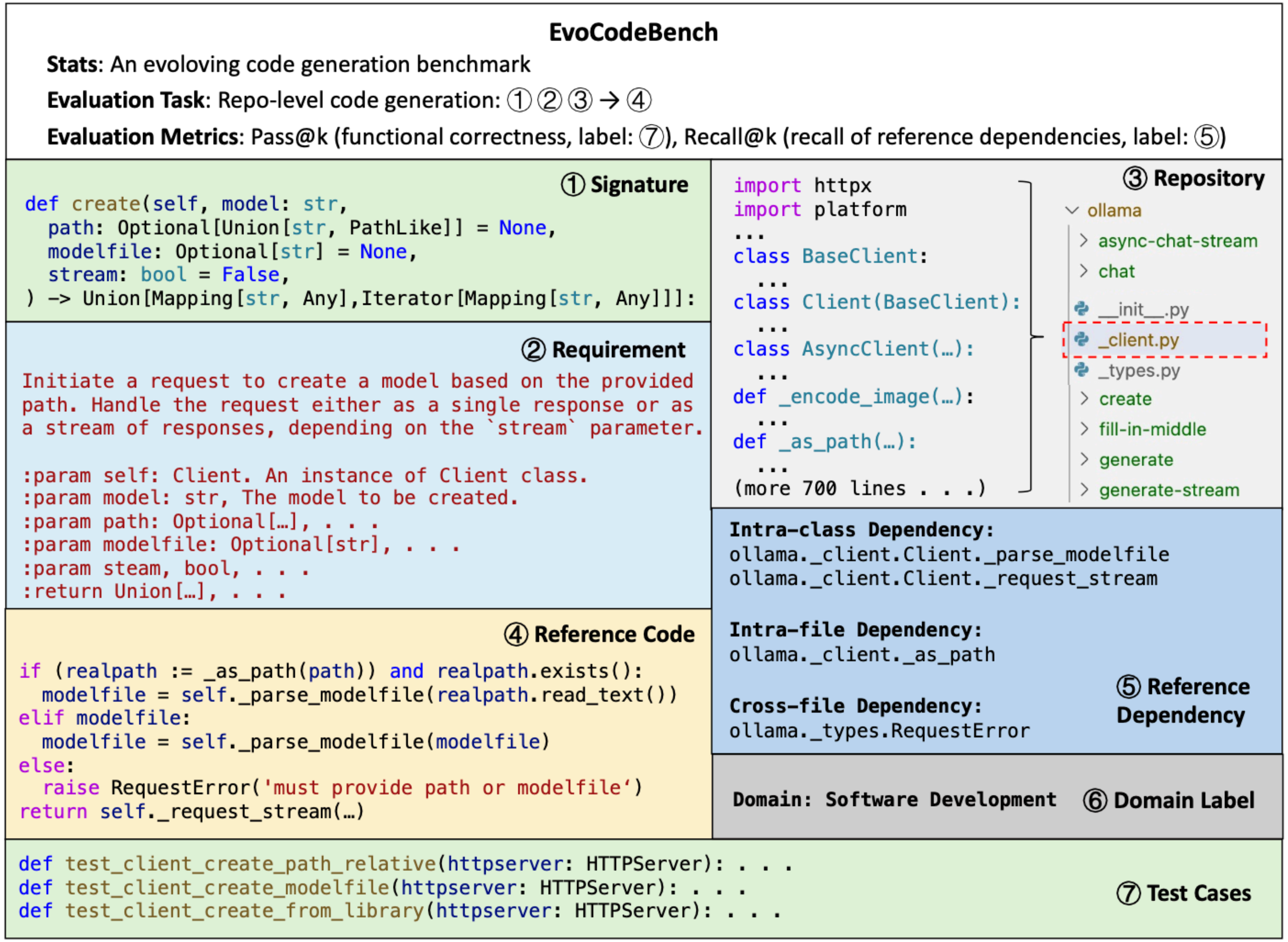
Introduction

- Evolving data
  - Dynamically updated every period (e.g., 6 months) to avoid data leakage
- A domain taxonomy and domain labels
  - Domain taxonomy consisting of 10 popular domains
- Domain-specific evaluations.
  - Domain-Specific Improvement (DSI) and define LLMs' comfort and strange domains

Introduction

- Data leakage (aka data contamination)
  - Training data of LLMs contains almost all open-source code repositories, existing benchmarks probably have data leakages. Researchers have to spend more effort to construct new benchmarks
- Lack of domain-specific evaluation
  - Compared to comprehensive coding abilities, developers are more concerned about the performance of LLMs in specific domains. However, existing benchmarks lack domain labels or fall into narrow domains. Besides, they ignore domain- specific evaluations and analyses.

Overview


EvoCodeBench
Stats: An evoloving code generation benchmark
Evaluation Task: Repo-level code generation: ①②③ → ④
Evaluation Metrics: Pass@k (functional correctness, label: ⑦), Recall@k (recall of reference dependencies, label: ⑤)
① Signature
def create(self, model: str,
path: Optional[Union[str, PathLike]] = None,
modelfile: Optional[str] = None,
stream: bool = False,
) -> Union[Mapping[str, Any],Iterator[Mapping[str, Any]]]:
② Requirement
Initiate a request to create a model based on the provided path. Handle the request either as a single response or as a stream of responses, depending on the `stream` parameter.
:param self: Client. An instance of Client class.
:param model: str, The model to be created.
:param path: Optional[…], . . .
:param modelfile: Optional[str], . . .
:param steam, bool, . . .
:return Union[…], . . .
③ Repository
import httpx
import platform
...
class BaseClient:
...
class Client(BaseClient):
...
class AsyncClient(…):
...
def _encode_image(…):
...
def _as_path(…):
...
(more 700 lines . . .)
ollama
async-chat-stream
chat
__init__.py
_client.py
_types.py
create
fill-in-middle
generate
generate-stream
④ Reference Code
if (realpath := _as_path(path)) and realpath.exists():
modelfile = self._parse_modelfile(realpath.read_text())
elif modelfile:
modelfile = self._parse_modelfile(modelfile)
else:
raise RequestError('must provide path or modelfile')
return self._request_stream(…)
Intra-class Dependency:
ollama._client.Client._parse_modelfile
ollama._client.Client._request_stream
Intra-file Dependency:
ollama._client._as_path
Cross-file Dependency:
ollama._types.RequestError
⑤ Reference Dependency
Domain: Software Development
⑥ Domain Label
def test_client_create_path_relative(httpserver: HTTPServer): . . .
def test_client_create_modelfile(httpserver: HTTPServer): . . .
def test_client_create_from_library(httpserver: HTTPServer): . . .
⑦ Test Cases

Task and Metrics

- EvoCodeBench evaluates LLMs in repo-level code generation. This task simulates the developers' coding process in a working repository.

$$\text{Pass@}k := \mathop{\mathbb{E}}_{\text{Requirements}} \left[1 - \frac{\binom{n-c}{k}}{\binom{n}{k}} \right]$$

Functional Correctness

$$\text{Recall@}k := \mathop{\mathbb{E}}_{\text{Requirements}} \left[\max_{i \in [1,k]} \frac{|\mathbb{R} \cap \mathbb{P}_i|}{|\mathbb{R}|} \right]$$

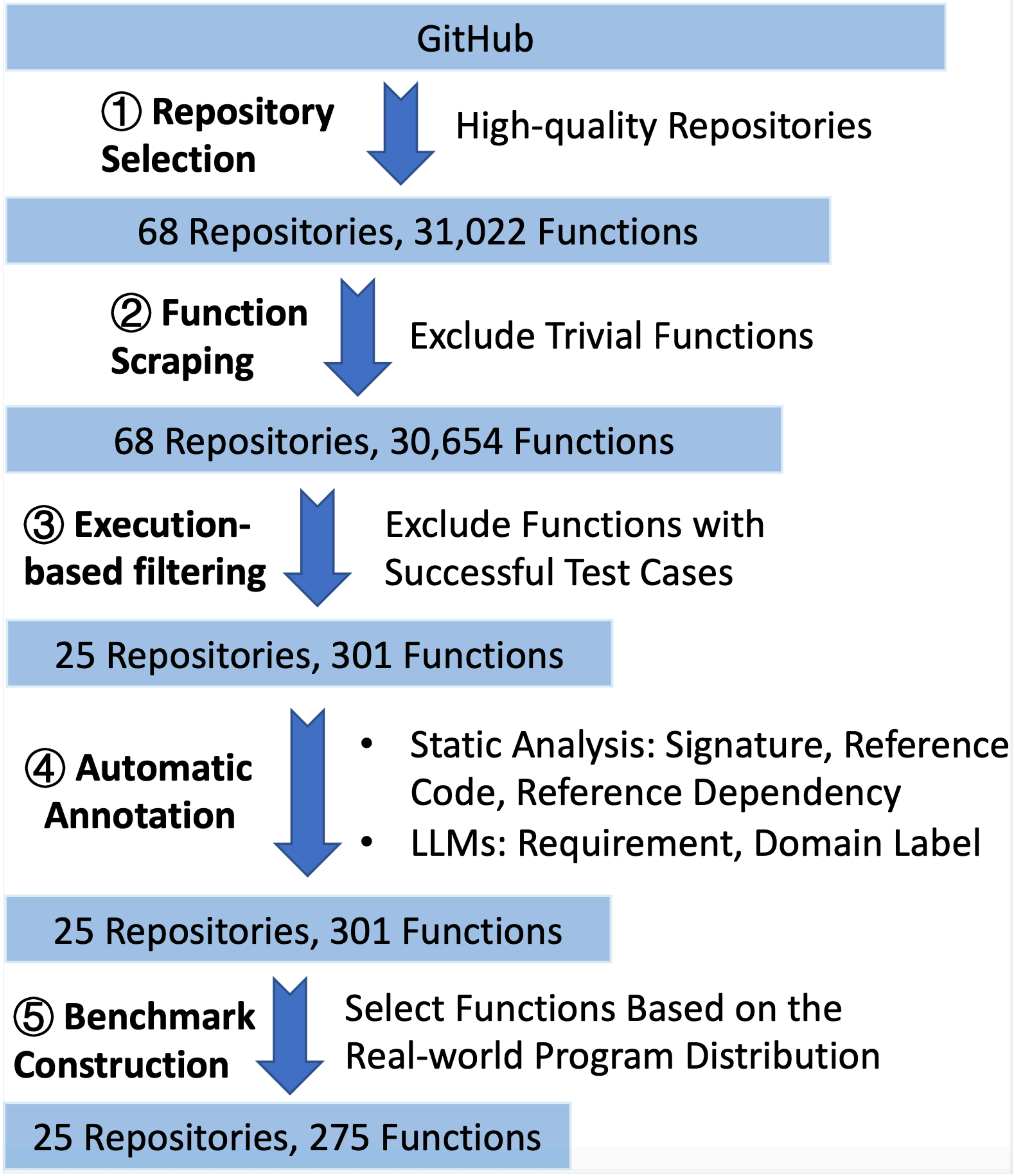
Recall of Reference Dependency

Data Construction

- Stage I: Repository selection and function scraping
- Stage II: Execution-based filtering
- Stage III: Automatic annotations
- Stage IV: Benchmark Construction

Table 1: The domain distribution of EvoCodeBench-2403.

Domain	Count
Scientific Engineering	120
Software Development	50
Multimedia	32
Database	18
System	17
Internet	15
Text Processing	12
Communications	8
Utilities	2
Security	1



Advantages

- Latest repositories to avoid data leakage (October 2023 - March 2024)
- Diverse domains
- High data quality

Table 1: The domain distribution of EvoCodeBench-2403.

Domain	Count
Scientific Engineering	120
Software Development	50
Multimedia	32
Database	18
System	17
Internet	15
Text Processing	12
Communications	8
Utilities	2
Security	1

Benchmark	Code Distribution				Dependency Distribution		Annotation
	#Repo.	#Sample	SA	Non-SA	#Type	#Avg.	
CoNaLa [28]	–	500	100%	0%	0	0	NL, Code
HumanEval [3]	–	164	100%	0%	0	0	NL, Code
MBPP [1]	–	974	100%	0%	0	0	NL, Code
APPS [11]	–	5,000	100%	0%	0	0	NL, Code
PandasEval [30]	–	101	100%	0%	0	0	NL, Code
NumpyEval [30]	–	101	100%	0%	0	0	NL, Code
AixBench [17]	–	175	100%	0%	0	0	NL, Code
ClassEval [7]	–	100	100%	0%	0	0	NL, Code, Depend. Name
Concode [12]	–	2,000	20%	80%	1	1.23	NL, Code
CoderEval [29]	43	230	36%	64%	3	1.73	NL, Code, Depend. Name
DevEval [16]	117	1,874	27%	73%	3	3.41	NL, Code, Depend, Repo
EvoCodeBench-2403	25	275	27%	73%	3	3.46	NL, Code, Depend Repo, Domain
500 Real Repositories	500	1M+	27%	73%	3	3.22	–

Experiment Setting

- We use the latest data leakage detection approach CDD to check EvoCodeBench-2403. CDD can detect whether LLMs have been trained on specific benchmarks and their variants.
- Compared to a mainstream benchmark - HumanEval, the leakage rate of EvoCodeBench- 2403 drops significantly to less than 3%.
- We think that EvoCodeBench-2403 is leakage-free and can provide trustworthy evaluations in repo-level code generation.

Table 3: The results of data leakage detection.

Benchmark	LLMs	Leak Ratio (%) ↓
HumanEval	gpt-3.5	**41.47**
EvoCodeBench-2403	gpt-4	2.18
	gpt-3.5	1.75
	DeepSeek Coder-33B	1.88
	DeepSeek Coder-7B	1.82
	StarCoder 2-15B	1.45
	StarCoder 2-7B	1.09
	CodeLLaMa-13B	0.82
	CodeLLaMa-7B	0.73

Repo-level Code Generation

- ❶ Without context. We ignore contexts and directly generate the code based on requirements and signatures.

- ❷ Local File (Completion). The local file denotes the code file where the reference code is in. This setting simulates the scenario where developers continue to write code at the end of a file. Besides the requirements and signatures, LLMs can access code contexts above the reference code in the local file.

- ❸ Local File (Infilling). This setting simulates the scenario where developers infill code in the middle of a file. Besides requirements and signatures, LLMs can see the code contexts above and below the reference code in the local file.

Repo-level Code Generation

Table 4: Pass@k and Recall@k of LLMs on EvoCodeBench-2403. Bold and underlined data indicate top-1 and top-2 results, respectively.

LLMs	Size	Pass@1	Pass@3	Pass@5	Pass@10	Recall@1	Recall@3	Recall@5	Recall@10
Local File (Infilling)									
gpt-4	N/A	**20.73**	**23.03**	**24.11**	**25.34**	68.24	70.63	72.05	73.52
gpt-3.5	N/A	17.82	21.78	23.06	24.46	61.94	68.13	69.69	70.85
DeepSeek Coder	33B	<u>19.64</u>	<u>22.78</u>	<u>24.29</u>	<u>26.01</u>	**71.46**	**79.93**	**82.11**	**86.25**
DeepSeek Coder	6.7B	17.82	21.02	22.40	23.97	<u>69.58</u>	<u>74.04</u>	<u>78.00</u>	<u>83.22</u>
StarCoder 2	15B	15.27	17.54	18.63	20.09	50.90	53.29	55.89	61.76
StarCoder 2	7B	14.91	17.29	18.63	19.86	56.35	60.59	63.74	74.20
Local File (Completion)									
gpt-4	N/A	**17.45**	**19.65**	**20.80**	**22.41**	63.49	68.67	70.00	72.07
gpt-3.5	N/A	<u>15.64</u>	17.29	18.21	19.36	61.44	66.25	66.82	69.89
DeepSeek Coder	33B	14.18	<u>17.57</u>	18.66	19.95	<u>66.90</u>	**72.83**	74.40	80.02
DeepSeek Coder	6.7B	13.45	17.10	<u>18.81</u>	<u>21.07</u>	65.76	<u>72.32</u>	<u>75.61</u>	78.45
StarCoder 2	15B	13.82	15.44	17.84	19.59	**68.55**	71.37	74.76	77.70
StarCoder 2	7B	13.45	15.15	16.18	17.65	62.93	69.85	73.54	78.40
CodeLLaMa	13B	12.73	15.78	16.86	18.19	63.34	71.26	**76.43**	<u>80.11</u>
CodeLLaMa	7B	12.73	15.33	16.00	16.93	63.33	69.79	71.91	76.50
Without Context									
gpt-4	N/A	**7.27**	**10.05**	**10.70**	<u>11.49</u>	21.58	23.93	25.69	26.23
gpt-3.5	N/A	6.55	7.85	8.28	8.73	21.66	24.31	24.77	25.40
DeepSeek Coder	33B	<u>6.91</u>	<u>8.92</u>	9.79	11.03	**27.67**	**32.73**	<u>34.92</u>	<u>37.76</u>
DeepSeek Coder	6.7B	5.82	8.56	9.67	11.26	25.89	32.06	**35.59**	**38.33**
StarCoder 2	15B	6.18	8.77	<u>9.95</u>	**11.53**	24.03	29.86	33.62	36.91
StarCoder 2	7B	5.82	6.72	7.43	8.62	<u>27.39</u>	<u>32.60</u>	34.88	36.81
CodeLLaMa	13B	5.45	7.38	8.37	9.95	25.52	31.28	33.66	36.36
CodeLLaMa	7B	5.45	6.94	7.75	9.03	26.97	31.17	34.08	36.82

Case Study

- Compared to previous benchmarks, these LLMs' performance in EvoCodeBench-2403 drops dramatically.
- LLMs benefit from more code contexts in repo-level code generation.
    - Uniquely successful case in the Local File (Completion) setting. Without context, gpt-4 fabricated a non-existent field as cache directories, generating the incorrect code. After introducing the local file, gpt-4 successfully invokes relevant functions and generates the correct code.

Code Contexts (above):

```
import concurrent
import json
def _get_num_nodes() -> int:
                          . . .
def _get_cache_dir(name: Optional[str] = None):
                          . . .
def _get_cache_data_dir(name: Optional[str] = None):
                          . . .
```

Input Code:

```
def _cleanup_cache(self) -> None:
  """ Clean up cache directories by removing them if
they exist to prevent issues from corrupted files
from previous runs, and then recreates these
directories to ensure they are available for use.
  :param self: DataProcessor. ...
  :return: No return values.
  """
```

gpt-4-turbo-1106 (without context):

```
for cache_dir in self.cache_directories:
# Check if the cache directory exists
  if os.path.exists(cache_dir):
    . . .
```

Non-existent attribute

gpt-4-turbo-1106 (local completion):

```
# Define the cache directories
cache_data_dir = _get_cache_data_dir()
if os.path.exists(cache_data_dir):
  . . .
```

APIs defined in contexts

Domain Lessons

- EvoCodeBench shows superior LLMs in specific domains.
- EvoCodeBench uncovers the comfort domains and strange domains of specific LLMs.
  - GPT-4 has the most comfort domains. Among all LLMs, only gpt-4 successfully solves some programming tasks. However, gpt-4 performs worse than others in the Internet domain.
  - StarCoder 2-15B unexpectedly performs well in the Database domain and even is comparable to gpt-4.
  - The potential reason for comfort and strange domains is that the pre-training data mix of LLMs is different. The finding can help model trainers analyze the shortcomings of existing LLMs and build more powerful code LLMs.

Table 6: The Domain-Specific Improvements (%) of LLMs in different domains. The comfort domains and strange domains are marked in bleu and red, respectively.

Domain	gpt-4	gpt-3.5	DeepSeek Coder 33B	DeepSeek Coder 6.7B	StarCoder 2 15B	StarCoder 2 7B	CodeLLaMa 13B	CodeLLaMa 7B
Database	10.21	10.21	-7.14	-7.14	-7.14	10.21	-7.15	-7.15
System	9.51	9.52	9.51	9.51	-11.42	-11.42	-42.84	9.52
Software Development	23.81	23.81	-21.43	23.81	-66.67	5.71	-21.43	-21.43
Internet	-28.59	7.15	7.15	7.15	7.15	-28.59	7.15	7.15
Scientific Engineering	26.55	11.90	11.90	-39.22	2.63	-8.63	-22.23	-8.63
Multimedia	32.14	4.75	-17.11	-17.11	4.75	4.75	4.75	-50.01
Text Processing	100.00	-100	-100	-100	-100	-100	-100	-100

Conclusion

- We introduce EvoCodeBench, an evolving code generation benchmark.
- EvoCodeBench is designed to address two limitations (i.e., data leakage and lack of domain-specific evaluations).
- We design a programming domain taxonomy consisting of ten popular domains and annotate samples with domain labels.
- We conduct extensive experiments on EvoCodeBench and reveal the actual abilities of LLMs in real-world repositories. We also evaluate LLMs in different domains and discover their comfort and strange domains.
- In the future, we will continuously release new versions of EvoCodeBench and extend EvoCodeBench into other programming languages (e.g., Java and C++).