CodeQueries Supplementary Document

A Additional Details

A.1 Dataset statistics

Table 1 gives the dataset statistics according to the splits of the ETH Py150 Open dataset. We place an example derived from a Python file in the same split as the file. The Min/Max entries give the number of minimum/maximum examples over individual queries, whereas Total is the sum of examples across all queries.

Table 1: Dataset statistics.

		Train	Validation	Test
Positive	Min	34	2	14
	Max	11,490	1,249	6,439
	Total	**20,783**	**2,319**	**11,560**
Negative	Min	29	1	17
	Max	17,592	1,893	9,892
	Total	**31,676**	**3,464**	**17,473**

A.2 Query Definitions

The definition of queries used in the dataset can be referred on the CodeQL site - `https://codeql.github.com/codeql-query-help/python/`.

A.3 Query-wise Dataset Statistics

We report the query-wise statistics for multi-hop and single-hop queries, aggregated across all splits, in Table 2 and Table 3 respectively. We report the statistics for *All Examples*, *Positive* examples, and *Negative* examples. *Count* gives the number of examples. A single file may be part of examples of multiple queries. Each example in Table 2 and Table 3 corresponds to a query and file pair, whereas Table 1 tabulates the number of *unique* files in different splits of the dataset. We sort all the tables from here on by the descending order of the count of all examples. Under all examples, we give the average length of the input sequences in terms of sub-tokens. Here, the sub-tokenization is performed using the CuBERT vocabulary. For positive examples, we report the average number of answer (abbreviated as *Ans.*) spans and supporting fact (abbreviated as *SF*) spans. Note that the number of answer or supporting fact spans is zero for negative examples and are hence omitted. We highlight the minimum and maximum values per column in bold face.

Table 2: Query-wise statistics for the multi-hop queries.

Index	**Query Name**	**All Examples**		**Positive**			**Negative**
		Count	Avg. Length	Count	Avg. Ans. Spans	Avg. SF Spans	Count
Q1	Unused import	**48,555**	3037.87	**19,178**	2.1	**0**	**29,377**
Q2	Missing call to `__init__` during object initialization	1,115	6860.32	353	2.18	3.06	762
Q3	Use of the return value of a procedure	919	6514.05	348	1.67	1.02	571
Q4	Wrong number of arguments in a call	700	8266.05	272	1.61	1.12	428

Index	Query Name	All Examples		Positive			Negative
		Count	Avg. Length	Count	Avg. Ans. Spans	Avg. SF Spans	Count
Q5	`__eq__` not overridden when adding attributes	547	8429.84	500	1.56	**5.61**	**47**
Q6	Comparison using is when operands support `__eq__`	453	10136.81	151	2.05	**0**	302
Q7	Non-callable called	375	9362.16	118	2.23	1.84	257
Q8	Signature mismatch in overriding method	374	11245.87	127	**2.32**	1.32	247
Q9	`__init__` method calls overridden method	371	**11335.33**	176	1.31	4.44	195
Q10	`__iter__` method returns a non-iterator	266	9196.37	165	1.27	1.36	101
Q11	Conflicting attributes in base classes	255	8920.37	96	1.9	3.07	159
Q12	Flask app is run in debug mode	242	**1134.98**	123	**1.0**	**0**	119
Q13	Inconsistent equality and hashing	195	9964.23	100	1.21	1.21	95
Q14	Wrong number of arguments in a class instantiation	188	7608.82	**79**	1.46	0.96	109
Q15	Incomplete ordering	**153**	9628.29	80	1.09	1.43	73
Aggregate		54,708	3617.26	21,866	2.04	0.30	32,842

Table 3 gives the query-wise statistics for single-hop queries aggregated across all splits. The column headings have the same meaning as those of Table 2. We highlight the minimum and maximum values per column in bold face.

Table 3: Query-wise statistics for the single-hop queries.

Index	Query Name	All Examples		Positive			Negative
		Count	Avg. Length	Count	Avg. Ans. Spans	Avg. SF Spans	Count
Q16	Unused local variable	**22,711**	5399.66	**8,123**	2.53	0	**14,588**
Q17	Except block handles `BaseException`	14,893	5081.62	5,909	2.23	**0**	8,984
Q18	Variable defined multiple times	8,548	7147.93	2,596	2.58	1.94	5,952
Q19	Imprecise assert	6,699	**4089.02**	2,192	5.67	**0**	4,507
Q20	Unreachable code	4,146	8025.58	1,726	1.46	**0**	2,420
Q21	Testing equality to None	4,045	8100.94	1,408	2.27	**0**	2,637
Q22	First parameter of a method is not named `self`	2,357	8031.02	444	4.6	**0**	1,913
Q23	Module is imported with `import` and `import from`	1,918	5057.49	912	1.11	**0**	1,006
Q24	Unnecessary pass	1,812	7902.1	757	1.86	**0**	1,055
Q25	Module is imported more than once	953	5384.63	391	1.45	1.13	562
Q26	Comparison of constants	839	10276	61	**13.72**	**0**	778
Q27	Implicit string concatenation in a list	787	8942.5	237	2.35	**0**	550
Q28	Suspicious unused loop iteration variable	750	9927.05	317	1.36	**0**	433
Q29	Duplicate key in dict literal	675	8655.74	131	4.56	**4.37**	544
Q30	Unnecessary `else` clause in loop	606	9305.47	278	1.24	**0**	328
Q31	Redundant assignment	566	7991.59	231	1.46	**0**	335
Q32	First argument to super() is not enclosing class	560	5322.95	236	1.4	**0**	324
Q33	Import of deprecated module	500	5889.08	228	1.19	**0**	272
Q34	Nested loops with same variable	496	9117.3	222	1.26	1.14	274
Q35	Redundant comparison	425	10775.71	153	1.81	1.6	272

Index	Query Name	All Examples		Positive			Negative
		Count	Avg. Length	Count	Avg. Ans. Spans	Avg. SF Spans	Count
Q36	An assert statement has a side-effect	408	6800.28	109	3.12	**0**	299
Q37	`import *` may pollute namespace	397	5441.52	197	**1.02**	**0**	200
Q38	Constant in conditional expression or statement	377	9761.03	118	2.19	**0**	259
Q39	Comparison of identical values	358	9861.62	108	2.32	**0**	250
Q40	Illegal raise	342	7482.82	141	1.43	**0**	201
Q41	NotImplemented is not an Exception	340	6763.09	124	1.93	**0**	216
Q42	Unnecessary delete statement in function	309	8875.78	146	1.36	1.36	163
Q43	Deprecated slice method	285	10171.74	86	2.6	**0**	199
Q44	Insecure temporary file	249	6488.8	107	1.41	**0**	142
Q45	Modification of parameter with default	230	9112.3	88	1.61	1.23	142
Q46	Should use a `with` statement	204	6525.19	91	1.26	0.02	113
Q47	Use of `global` at module level	182	6614.24	72	1.69	**0**	110
Q48	Non-standard exception raised in special method	167	9277.62	65	1.58	0.14	102
Q49	Modification of dictionary returned by locals()	165	7130.5	65	1.51	**0**	100
Q50	Special method has incorrect signature	164	**11703.91**	56	1.98	1.48	108
Q51	Incomplete URL substring sanitization	154	5805.74	62	1.61	**0**	92
Q52	Unguarded next in generator	**131**	7526.71	**54**	1.52	**0**	**77**
Aggregate		78,748	6228.49	28,241	2.51	0.25	50,507

A.4 Query-wise Span Prediction Analysis

We performed the query-wise analysis of the predictions of the CuBERT-1K model for the Span prediction problem. Tables 4 and 5 give the query-wise results for the multi-hop and single-hop queries, respectively, with minimum and maximum results highlighted in boldface.

Among the multi-hop queries, on positive examples, the model struggles the most on the top-2 queries (Q9 and Q7) by the average number of tokens in examples (see Table 2), and the query Q12 has the highest exact match due to the simplicity of the query. On negative examples, the query Q8 works best at the cost of positive examples, and the worst performing query Q4 has the smallest number of negative examples (see Table 2).

Table 4: Query-wise results for the multi-hop queries.

Id	Query Name	Exact Match		
		All Examples	Positive	Negative
Q1	Unused import	78.30	52.10	95.35
Q2	Missing call to `__init__` during object initialization	88.06	85.45	90.67
Q3	Use of the return value of a procedure	75.32	50.34	96.49
Q4	`__eq__` not overridden when adding attributes	56.18	63.90	**11.90**
Q5	Wrong number of arguments in a call	48.97	14.81	76.30
Q6	Comparison using is when operands support `__eq__`	82.11	64.63	95.37
Q7	Signature mismatch in overriding method	51.58	2.97	92.50
Q8	Non-callable called	59.21	8.82	**100.00**
Q9	`__init__` method calls overridden method	**34.27**	**2.82**	65.28
Q10	Conflicting attributes in base classes	47.46	16.95	77.97
Q11	`__iter__` method returns a non-iterator	79.25	96.77	54.55

Id	Query Name	Exact Match		
		All Examples	Positive	Negative
Q12	Flask app is run in debug mode	**97.47**	**97.50**	97.44
Q13	Inconsistent equality and hashing	64.06	71.88	56.25
Q14	Wrong number of arguments in a class instantiation	48.61	23.53	71.05
Q15	Incomplete ordering	63.79	48.28	79.31
Aggregate		76.82	52.19	94.01

Among the single-hop queries, there are 6 queries with fewer than 100 positive examples (see Table 3). Of these, except for the query Q49, all others are among the worst performing. The queries Q36 and Q38 are simple and are the best performing on positive examples. On negative examples, the model achieves a very high exact match ($> 95\%$) for several queries; and the query Q42 has the lowest exact match.

Table 5: Query-wise results for the single-hop queries.

Id	Query Name	Exact Match		
		All Examples	Positive	Negative
Q16	Unused local variable	91.78	82.90	98.34
Q17	Except block handles `BaseException`	96.56	92.71	99.82
Q18	Imprecise assert	98.64	96.25	99.95
Q19	Variable defined multiple times	78.56	50.13	95.44
Q20	Testing equality to None	97.31	93.82	**100.00**
Q21	Unreachable code	78.07	56.04	96.68
Q22	First parameter of a method is not named `self`	99.93	99.86	**100.00**
Q23	Unnecessary pass	95.79	91.28	99.39
Q24	Module is imported with `import` and `import from`	79.85	67.67	91.24
Q25	Module is imported more than once	72.18	50.58	91.62
Q26	Comparison of constants	93.79	42.86	95.84
Q27	Implicit string concatenation in a list	83.17	62.60	98.31
Q28	Suspicious unused loop iteration variable	86.79	79.38	93.04
Q29	Duplicate key in dict literal	59.69	10.71	97.26
Q30	Unnecessary `else` clause in loop	86.92	74.53	99.07
Q31	First argument to super() is not enclosing class	95.65	92.98	98.28
Q32	Redundant assignment	91.24	83.53	97.25
Q33	An assert statement has a side-effect	84.52	62.90	98.92
Q34	Nested loops with same variable	53.53	31.71	73.86
Q35	Import of deprecated module	72.63	42.05	99.02
Q36	NotImplemented is not an Exception	**100.00**	**100.00**	**100.00**
Q37	Redundant comparison	56.91	12.99	89.42
Q38	Deprecated slice method	**100.00**	**100.00**	**100.00**
Q39	Constant in conditional expression or statement	88.31	67.92	99.01
Q40	Comparison of identical values	83.46	58.00	98.80
Q41	`import` $\star$ may pollute namespace	93.02	87.50	98.46
Q42	Unnecessary delete statement in function	72.63	77.55	**67.39**
Q43	Illegal raise	86.73	85.11	88.24
Q44	Insecure temporary file	95.65	90.91	**100.00**
Q45	Modification of parameter with default	60.00	**9.09**	92.31
Q46	Should use a `with` statement	95.00	92.50	97.50
Q47	Special method has incorrect signature	82.43	78.38	86.49
Q48	Non-standard exception raised in special method	95.31	96.88	93.75
Q49	Use of `global` at module level	92.06	79.17	100.00
Q50	Modification of dictionary returned by locals()	75.86	47.83	94.29
Q51	Incomplete URL substring sanitization	76.71	39.29	**100.00**
Q52	Unguarded next in generator	**42.86**	13.04	69.23
Aggregate		91.55	82.34	98.16

A.5 Statistics of Syntactic Patterns of Spans

In our dataset, the answer and supporting-fact spans cover various types of programming language constructs. Hence, in Table 6, we tabulate the number of spans in terms of syntactic patterns of Python constructs in decreasing order of their frequency in the combined data of all three splits. To find the pattern of a span, we have used `tree-sitter`[1] to get the closest ancestor node which encloses the tokens appearing in the span. Two special entries in the table are block and module. A *block* node can represent any block of code, i.e., a block of code, a function, a class. Sometimes the closest ancestor node is the root node of the source code, for those cases *module* node is used as a representative node.

Table 6: Statistics of syntactic patterns of spans.

Syntactic Pattern	Count	Syntactic Pattern	Count	Syntactic Pattern	Count
import statement	43,013	raise statement	375	module	56
assignment	32,422	function parameters	373	dictionary keys	47
call	15,978	assert statement	368	break statement	43
except clause	13,269	delete statement	358	while statement	43
function definition	8,937	if statement	243	argument list	34
non-boolean binary operator	5,319	sequence expressions	192	with statement	26
class attributes	2,844	identifier	186	parenthesized expression	14
class definition	2,882	decorator	138	boolean operator	13
block	2,331	print statement	126	elif clause	12
pass statement	1,451	global statement	125	expression list	12
string literal	1,279	list comprehension	101	lambda	11
for statement	1,164	subscript	72	conditional expression	8
concatenated string	558	not operator	71	yield	5
return statement	395	try statement	65	continue statement	3
				Aggregate	134,962

A.6 Comparison to Existing Datasets

Existing datasets for question-answering in the context of programming languages target comparatively simpler tasks of predicting binary yes/no answers to a question or range over a localized context (e.g., a source-code method). In contrast, in CodeQueries, a source-code file is annotated with the required spans for a code analysis query about semantic aspects of code. Such a dataset can be used to experiment with various methodologies in an extractive question-answering setting with file-level code context. We tabulate a brief comparison of existing datasets considered for question-answering tasks on source code in Table 7.

References

[1] Bansal, A., Eberhart, Z., Wu, L., and McMillan, C. A neural question answering system for basic questions about subroutines. In *28th IEEE International Conference on Software Analysis, Evolution and Reengineering*. IEEE, 2021.

[2] Huang, J., Tang, D., Shou, L., Gong, M., Xu, K., Jiang, D., Zhou, M., and Duan, N. Cosqa: 20, 000+ web queries for code search and question answering. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing, ACL/IJCNLP*. Association for Computational Linguistics, 2021.

[3] Lee, C., Seonwoo, Y., and Oh, A. CS1QA: A dataset for assisting code-based question answering in an introductory programming course. In *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pp. 2026–2040, 2022.

[1] `https://github.com/tree-sitter/py-tree-sitter`

Dataset	**Size (Language)**	**Task**	**Evaluation Criteria**	**Code Context**
CoSQA (2)	20,604 (Python)	To check relevance between a web query and a method.	MRR	Method
CodeQA (4)	119,778 (Java) 70,085 (Python)	To generate free-form answers for template-based questions curated from comments.	BLEU, ROUGE-L, METEOR, Exact Match, F1	Method
Bansal et. al. (1)	≈10880K (Java)	To answer template-based basic questions on method characteristics	User study	Method
CS1QA (3)	9,237 (Python)	To classify the question into pre-defined types, identify relevant source code lines and retrieve related questions	Accuracy, F1, Exact Match (line-level)	Method
CodeQueries (this work)	133,456 (Python) See Tables 2–3 for details.	To extract answer spans from a given code context in response to a code analysis query, and provide reasoning with supporting-fact spans.	Exact Match	File

Table 7: Comparison to existing datasets on question-answering over source code.

[4] Liu, C. and Wan, X. Codeqa: A question answering dataset for source code comprehension. In *Findings of the Association for Computational Linguistics: EMNLP*. Association for Computational Linguistics, 2021.