UniTSyn: A Large-Scale Dataset Capable of Enhancing the Prowess of Large Language Models for Program Testing

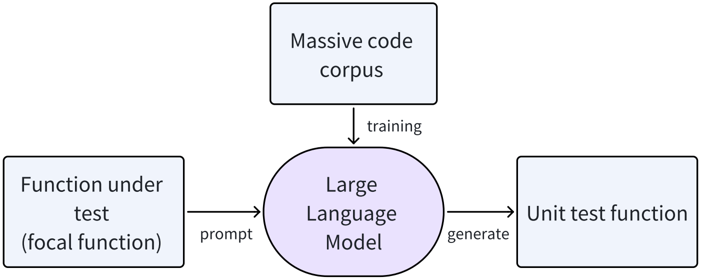
Yifeng He, Jiabo Huang, Yuyang Rong,
Yiwen Guo, Ethan Wang, Hao Chen

University of California, Daivs

September 24, 2024

UCDAVIS
COLLEGE OF ENGINEERING

LLM-based automatic unit test generation



Using an LLM to generate unit tests involves three steps:

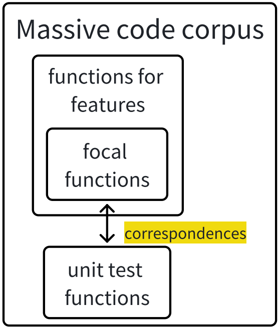
1. Train an LLM with a massive code corpus.
2. Prompt the LLM with focal function.
3. Let the LLM generate a unit test function.

Test generation is challenging for LLM

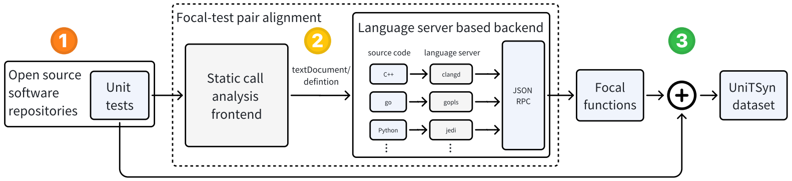
Unit test functions and their focal functions have:

1. Different representations.
2. Fundamental correspondences.

Therefore, a specialized dataset with **aligned focal-test pairs** is essential for LLM-based unit test generation.

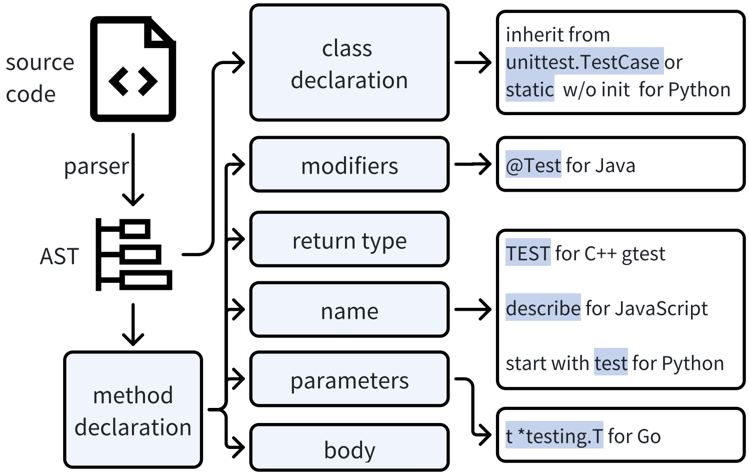


UniTSyn: a multilingual dataset with function-level focal-test alignment



1. We download open-source software repositories and extract their unit tests.
2. We use static analysis to identify the call to their focal functions and use the language server protocol to get the location of the focal function definition.
3. We store the aligned function-level focal-test pairs as training data.

Test function identification

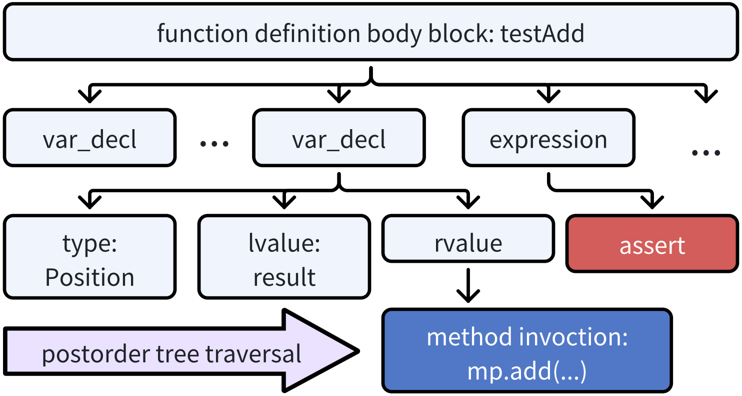


We design a static analysis algorithm to identify test functions across different programming languages.

1. Our algorithm traverses Abstract Syntax Trees (ASTs) to locate test functions by:
    1. heuristics: check function names (e.g., functions containing "test").
    2. language-specific features: use language-specific syntax, like Java's @Test modifier in JUnit.
2. Our dataset construction framework provides a language-agnostic interface to check for test functions using callback hooks.

This framework supports new languages by adding custom hooks for test function identification.

Focal function call analysis



Then we identify the focal function call within a unit test function.

1. We follow TeCo's [1] heuristic: Select the last function call before the first assertion as the focal function.
2. Our algorithm traverses the AST using a post-order method to detect the correct function call within the assertion.

Our design is

- extensible to multiple languages with minimal changes by adding one extra function to the analysis.
- applicable across different languages through a unified approach using the post-order tree traversal technique.

Dataset statistics

Table 1: Dataset statistics.
Framework: static analysis for test extraction
#Proj: number of projects found on GitHub for each language
#Pairs: number of focal-test pairs collected for each language

Language	Framework	#Proj	#Pairs
Python	unittest, pytest	43 848	1 218 311
Java	JUnit	25 488	1 097 518
Go	testing	38 097	361 075
C++	GoogleTest	20 090	25 513
JavaScript	MochaJS	17 621	13 293

Datasets comparison

Table 2: Datasets comparison.
#Proj: number of software projects in the dataset
#Lang: number of programming languages in the dataset
Unit Test: if the dataset specifically mines testing code
Alignment: the level of alignment between testing code (if exists) and code to be tested.

	The Stack [2]	CAT-LM [1]	TeCo [3]	UniTSyn (ours)
#Proj	137.36M	197 730	1270	246 194
#Lang	30	2	1	5+
Unit Test	✗	✓	✓	✓
Alignment	✗	file	function	function

Research questions

1. How accurate are the test cases generated by LLMs?
2. How many of the generated tests are complete?
3. Is it necessary to train LLMs with pairwise focal and test functions?
4. What are the effects of training with multilingual testing code?

Accuracy of generated test cases

Table 3: Accuracy of tests generated by LLMs. The best results are highlighted in bold.
#Params: the size of models
†: the models are intended for test generation.

Model	#Params	Py	C++	Java	JS	Go	Avg
CodeT5p	770M	30.6	33.7	26.9	37.1	32.9	32.2
CodeGen2	1.0B	34.0	40.7	24.1	30.5	36.1	33.1
WizardCoder	1.0B	36.8	43.9	28.7	31.3	47.7	37.7
InCoder	1.3B	34.2	33.5	22.6	24.4	31.5	29.2
SantaCoder	1.1B	36.2	34.7	36.5	30.6	31.5	33.9
CAT-LM†	2.7B	37.5	31.6	34.4	29.2	36.9	33.9
UniTester† (Ours)	1.1B	**52.5**	**55.1**	**48.8**	**41.7**	**59.7**	**51.5**

Coverage of generated unit tests on the focal function

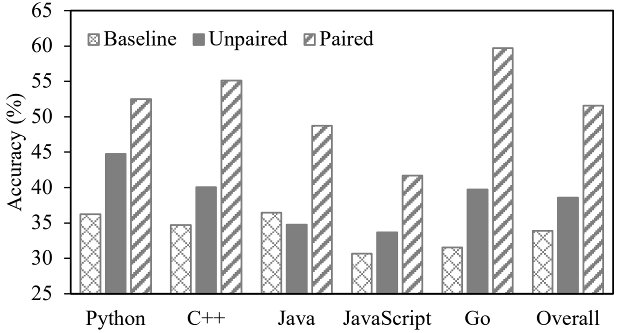
Table 4: Completeness of LLM-generated tests.
#Params: size of the model. #Pass: percentage of tests for the 164 tasks that can be executed without errors.
Line, Stmt: average line and statement coverage, respectively.
†: the model is intended for test generation.

Model	Python		C++		Java		Javascript		Go	
	#Pass	Line	#Pass	Line	#Pass	Line	#Pass	Line	#Pass	Stmt
CodeT5p	10.0	5.72	0.7	0.43	40.3	4.22	4.9	2.07	1.7	0.73
CodeGen2	4.1	2.41	11.6	7.07	52.3	5.12	48.5	27.65	19.2	10.99
WizardCoder	16.1	9.39	3.7	2.24	47.7	5.62	9.2	5.50	0.7	0.42
InCoder	3.0	1.76	0.0	0.00	15.0	1.54	0.5	0.29	1.3	0.78
SantaCoder	4.5	2.62	4.9	2.99	50.1	4.74	5.9	3.53	0.7	0.43
CAT-LM†	35.9	19.51	0.0	0.00	0.9	0.07	9.2	4.53	0.0	0.00
UniTester† (Ours)	41.2	20.71	28.1	13.39	103.1	10.78	53.3	27.59	36.0	12.39

RQ 1 & 2

RQ1: How accurate are the test cases generated by LLMs?
RQ2: How many of the generated tests are complete?
Our model trained on our dataset achieves the best assertion accuracy and branch/statement coverage.

Is it necessary to train LLMs with paired focal and test functions?

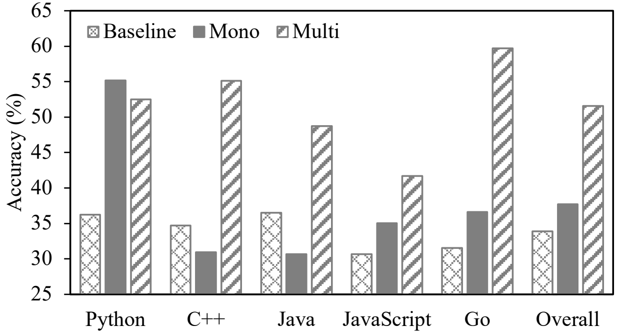


Impact of pairing test and focal functions.
Baseline: the SantaCoder model, not trained with our data.
Unpaired: trained with decoupled test and focal functions.
Paired: UniTester trained with focal-test pairs.

RQ 3

RQ3: Is it necessary to train LLMs with pairwise focal and test functions?
A: Training with function-level aligned focal-test pairs increases assertion accuracy in all five languages.

What are the effects of training with multilingual testing code?



Effects of training with multilingual testing code.
Baseline: the SantaCoder model, not trained with our data.
Mono: monolingual model trained with solely Python data.
Multi: multilingual models trained jointly with five languages.

RQ 4

RQ4: What are the effects of training with multilingual testing code?
A: For Python, the monolingual model demonstrated superior capability in assertion accuracy. For other languages with stricter syntax, the multilingual model achieves better results.

[1] P. Nie, R. Banerjee, J. J. Li, R. J. Mooney, and M. Gligoric, "Learning deep semantics for test completion," in *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*, pp. 2111–2123, IEEE, 2023.

[2] D. Kocetkov, R. Li, L. B. allal, J. LI, C. Mou, Y. Jernite, M. Mitchell, C. M. Ferrandis, S. Hughes, T. Wolf, D. Bahdanau, L. V. Werra, and H. de Vries, "The stack: 3 TB of permissively licensed source code," *Transactions on Machine Learning Research*, 2023.

[3] N. Rao, K. Jain, U. Alon, C. Le Goues, and V. J. Hellendoorn, "Cat-lm training language models on aligned code and tests," in *2023 38th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pp. 409–420, IEEE, 2023.

Thanks For Your Attention!

Any questions?