

# UniversalPython - A Multilingual Python Programming Language\*

\* Note: This draft was submitted in partial fulfillment of the requirements for the course "Research Methodology" at FAST-NUCES

Saad Ahmed Bazaz  
*Software Engineering and Product Design*  
Grayhat  
Islamabad, Pakistan  
bazaz@grayhat.studio

**Abstract—**

**Index Terms—computer science, computer science education, multilingual, programming language, internationalization, python, machine translation**

## I. INTRODUCTION

Students face difficulty in learning coding in a language other than their native one [1]

Many non-English programming languages [2]

## II. RELATED WORK

High-level computer programming languages have been predominantly in the English language, ever since the first widespread high-level programming language, FORTRAN [3], which had an English instruction set. It is interesting to note, however, that the first high-level programming language recorded in history, the "Plankalkül" of German civil engineer Konrad Zuse [4], often considered a forerunner of today's programming languages [5].

### A. Learning in your own language

Studies have shown that students in K-12 are able to learn better, in general studies, in their localized language. [6] Some successful attempts have been made across the globe in making non-dominant languages part of the curriculum, to make classrooms inclusive, preserve culture, and improve understanding. [7]

In certain developing parts of the world, language can present a barrier to entry, where young people in rural settings may dropout of school due to education being in a language other than their own [8], and so literacy classes in the mother tongue can be of great help in re-integrating these youth into the school system. Girls in particular experience significant positive effects of local-language learning contexts, including better achievement, better self-image as learners and longer retention in school [9].

Computing education is particularly challenging, due to English being the predominant language for programming. Efforts have been made to localize computing education content to local, with the example of a Qatari middle school curriculum [10], which have shown promising results.

Recent tests have shown that, using longitudinal data drawn from five countries and over 15,000 users of Scratch, a large informal learning community, novice users who code with their programming language keywords and environment localized into their home countries' primary language demonstrate new programming concepts at a faster rate than users from the same countries whose interface is in English. [11] This presents the opportunity to explore a form of universal design [12] by internationalizing (also called localizing) code examples helps everyone learn to write more robust code. One should also be able to "flip the language bit" and use those ideas to instead help native English speakers learn a foreign language within the context of their normal programming activities. [13]

### B. Non-English, monolingual programming languages

One of the earliest examples of a programming language dedicated to localized education dates to 1975 during the USSR era, when a Russian programming language called Robic [14], was developed for primary school education (8-11 year-old children). It was later modified to be included in a software system called "Schoolgirl" for the computer Agat. The language uses syntax based on the Russian vocabulary.

Various languages have been built since then, targeting a single language (mono-lingual) other than English. A Yoruba-based programming language [15], and Kalaam [16], a programming language in Hindi, are examples of such. However, such projects often die out due to lack of community engagement and interest, and cannot gather enough attention as there is not much practical use for the learner other than learning programming constructs, and because the language eventually becomes "another syntax to learn".

A for-fun programming language in Ancient Chinese [17] took the Internet by storm upon its release, prompting hundreds of thousands of users to create interesting programs in it.

More recently, an Arabic programming language, Alif [18], sports a Pythonic syntax, while itself being written in C/C++. While it benefits from aligning itself with Python, making it familiar enough for programmers to

take on for fun and for learning, it struggles to catch up with the developments happening in Python itself, simultaneously.

### C. Multilingual programming languages

There have been multiple attempts at multilingual programming languages. These are languages built from the ground up to be simple, many of them using just symbols and illustrations to explain programming to novices. They also make use of labelling for these symbols, which are then localized for audiences. Most of such programming languages are for educational purposes only.

Hedy [19] is a notable example, which is a mostly-textual focused programming language and environment focused on teaching novices how to code.

A famous example in this space is Scratch [20], which allows learners to visually program in easy, understandable steps.

Such attempts are great for learning, yet require a large amount of funding for maintenance of the entire ecosystem. Also, these tools and programming languages are mostly restricted to basic operations. Although, some content creators like *Griffpatch* on YouTube [21] have been able to produce impressive games with the tool.

### D. Localizing existing programming languages

PseuToPy [22] proposes an intermediate pseudo language to communicate with the Python interpreter. This approach adds complexity for users due to potential inconsistencies in machine translation. Additionally, PsueToPy lacks a standard structure, making it difficult to transition between languages.

A JavaScript dialect in Urdu, UrduScript [23], aims to "... make programming more accessible to beginners from South Asia" [24]

Another team took the approach of editing the Python source code to accommodate Chinese symbols [6]. However, this method is difficult to implement and maintain across different versions and languages.

Overall issues: Lacking open-source libraries, and limited scalability to other human languages.

### E. Thin wrappers

The challenges faced by multilingual programming is summarized well in [25],

The Python language has been more welcoming to a broader range of Unicode, allowing for variable names in other languages, which not only interpret correctly, but are also correctly referenceable. [26]

- Star Projects: - - legesher [27] - Universal Python [28]

## III. DESIGN OF UNIVERSALPYTHON

We propose a simple framework as shown in Figure 1, which is a wrapper around the Python interpreter. This is a transpiler, that is, a source-to-source compiler, which is able to translate a higher-level programming language (e.g. Urdu Python, Chinese Python, Hindi Python, Arabic

Python, etcetera) to the underlying language (in this case, just standard Python).

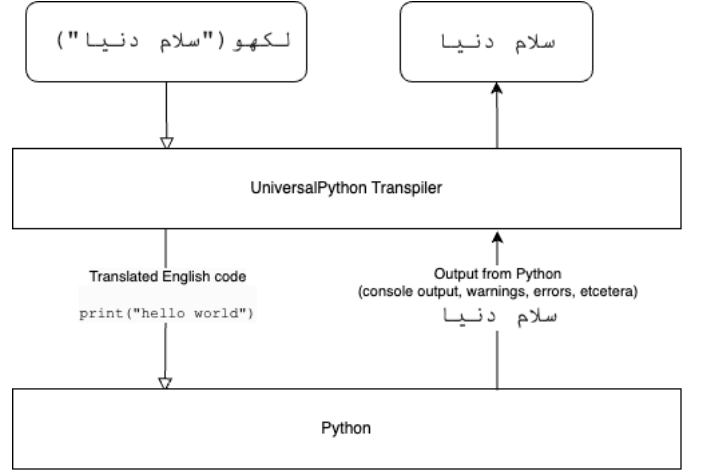


Figure 1: A high-level abstraction of how the UniversalPython transpiler works.

For example, if the user writes the following code in Urdu<sup>1</sup>:

```

۲ = کچھ
اگر کچھ == ۱:
    لکھو (\"سلام دنیا\")
ورنہ:
    لکھو (\"خدا حافظ\")

```

This is passed to “UniversalPython”, which first translates the code to English using Lexical Analysis and Parsing with the PLY library. To do this, it first loads the Urdu dictionary, which is a YAML file containing mappings from Urdu to English. This dictionary is a mapping of each Urdu word to a Python keyword. Here is an example of such a dictionary: In PLY, we have the option to reserve some keywords so that the library automatically tokenizes them. We set a Grammar Rule which, whenever a reserved keyword (i.e. a word which is present as a key in a language dictionary) is tokenized, simply looks for the token in the language dictionary (key) and replaces it with the corresponding English keyword (value). We also set a Grammar Rule to ignore all content within double quotes or single quotes (i.e. strings and docstring) and to ignore content in comments (which start with a #). We achieve this using Regular Expressions. Urdu numbers also lie on the Unicode scale. • (or roman 0) is at 1776, while ۹ (or roman 9) is at 1785.

Keeping this in mind, we define a Regular Expression which detects any symbols equal to or between this range, 1776 - 1785. These are the Urdu digits. We then use the same language dictionary to translate these numbers into

<sup>1</sup>Throughout this paper, we take the example of Urdu (اردو), the national language of Pakistan (the author’s home country), and a traditional Right-to-left language, as an example translation language.

roman numerals. For example, ۵ becomes 5, ۹۰ becomes 90, ۱۰ becomes 10, ۲۰۲۵ becomes 2025. Furthermore, we also replace all periods ( . ) and commas ( , ), as these look different in Urdu than they do in English.

The rest of the code remains untouched. Whether it is a symbol like :, ;, etc, or even if it causes the lexer to crash, we ignore it so we can preserve the original structure of the code as much as possible. It is not required to translate such symbols and errors anyway; They are meant to be handled by Python, not UniversalPython. Back to our initial example code, our transpiler detects and replaces it with print, اگر is replaced with if, ورنہ is replaced with else, and all Arabic digits are replaced with Roman digits. Hence the translated code would be:

---

```
khchh = 2
if khchh == 1:
    print ("دنیا سلام")
else:
    print ("ہاںظ خدا")
```

---

The above code is essentially vanilla Python code which can now simply be executed by the Python interpreter. The Urdu variable name was auto-translated to English using the unicode library, however, even does not present any issue to Python, as from Python 3.0 onwards, Unicode is fully supported. So the above code is passed on to the Python interpreter. The Python interpreter outputs some response; it can be some print statements which the user entered, compiler/interpreter warnings, and/or errors. This response is passed up to UniversalPython, where again it is tokenized to replace keywords in case of error messages. In our example, since there are no errors, it simply outputs the response as-is. So the response would be:

سلام دنیا

To demonstrate the ease by which plugins can be made for UniversalPython, we made a wrapper for the IPython kernel in which we imported UniversalPython as a package, and processed the code (i.e. translated it from Urdu to English) before it was passed onto IPython. This way, we overrode the functions and achieved a working kernel for UniversalPython. It works line-by-line while maintaining the program memory. This also gave us a visual interface for the language to test it thoroughly.

Table I: Language dictionary for periods and commas

Key	Value
.	۔
,	،

The Urdu dictionary is a YAML file containing mappings from Urdu to English keywords, is used for translation. The PLY library allows reserving keywords for automatic tokenization and replacement with their English equivalents. Additionally, grammar rules are set

to ignore content within quotes, comments, and Urdu numbers (which lie on the Unicode scale).

Python (original)	Urdu (اردو)
'print'	لکھو
'if'	اگر
'elif'	ورنہ
'else'	ورنہ
'while'	جب تک
'for'	جو
'in'	اندر
'input'	داخلہ
'break'	توڑ
'continue'	جاری
'pass'	گزر
'True'	حق
'False'	باطل

Table II: Comparison of Python Keywords and their Urdu Equivalents

## IV. EXPERIMENTATION

### A. Evaluation metrics

We propose the following metrics which UniversalPython should meet in order than it is considered effective:

- 1) Programs which work in Python, should work in UniversalPython.
- 2) UniversalPython should operate as a reasonable speed which at least does not disturb the programmer.
- 3) UniversalPython should be able to translate from one non-English language to another
- 4) A comparison should be made against other existing non-English, monolingual programming languages and multilingual programming languages
- 5) A user experience test should be conducted to find out user acceptability towards a language in their native tongue.

### B. Benchmarks with Python

**Time:** Check the execution time / performance of Python vs UniversalPython. We use a benchmarking tool called hyperfine which runs each program multiple times to produce a mean execution time.

**Conversion:** Convert simple programs from English Python to UniversalPython, and test if they still work. Our testing mechanism for the above is as below:

- 1) We take an existing Python program, lets say multiplication.py.
- 2) Run the UniversalPython system in reverse. This will flip the language dictionary and generate an Urdu version of the program (all the keywords would be translated from English to Urdu). Save it as multiplication.ur.py.
- 3) Run multiplication.ur.py using UniversalPython, and multiplication.py using Python.

If both give the same output on the terminal then no data loss has occurred; Both Urdu program and English program output the same message. Hence we can say

that the program has safely converted from English to UniversalPython and vice versa without breaking the code or changing the logic.

We take simple algorithmic programs from TheAlgorithms/Python [29], a repository containing implementations of well-known algorithms, in the Python programming language. We run a loop over them and run the above algorithm on each to test.

Our framework allows easy support for UniversalPython plugins in existing Python IDEs. It acts as a bridge, translating Urdu code into English before passing it to the IDE.

## V. RESULTS

### A. Benchmarking Python and UniversalPython code

We evaluate the results of the experiment mentioned above in the following two ways: Execution Success Rate. After running the experiment, we were able to achieve the following results.

“Table III” describes our results over 110 simple Python programs present in TheAlgorithms/Python [14] maths implementation. We reported that 98% of simple programs can be automatically converted into Urdu Python without breaking the code. Amongst the programs that failed, the reason was due to a problem in generating the Urdu version of an English program (i.e. running UniversalPython in reverse) - it cannot distinguish between “is” and “==”, as they have the same purpose in English Python. Execution Time Upon recording the execution time summary of the experiment we concluded that UniversalPython performed better with simpler and less wordy programs, whereas for more wordy and lengthy algorithms, python performed better in terms of execution time. However, it is well within the range of acceptable speed of a programming language. Some of the top differences in performances of some algorithms can be seen in the tables “Table IV” and “Table V”. Table IV: Execution times of functions where our transpiler performed well.

### B. Trying out different languages and translating between them

It is possible to make this framework generic enough to span multiple languages. We managed to make three variants; Urdu, Chinese and Hindi, to demonstrate UniversalPython’s capability to extend multiple languages.

We were then able to translate between languages, e.g. Urdu to Hindi (as shown in Figure 2), by using English as an *intermediate language*.

### C. Making the Jupyter Kernel

To demonstrate the ease by which plugins can be made for UniversalPython, we made a wrapper for the IPython kernel in which we imported UniversalPython as a package, and processed the code (i.e. translated it from Urdu to English) before it was passed onto IPython. This way, we overrode the `do_execute` and `do_complete` functions

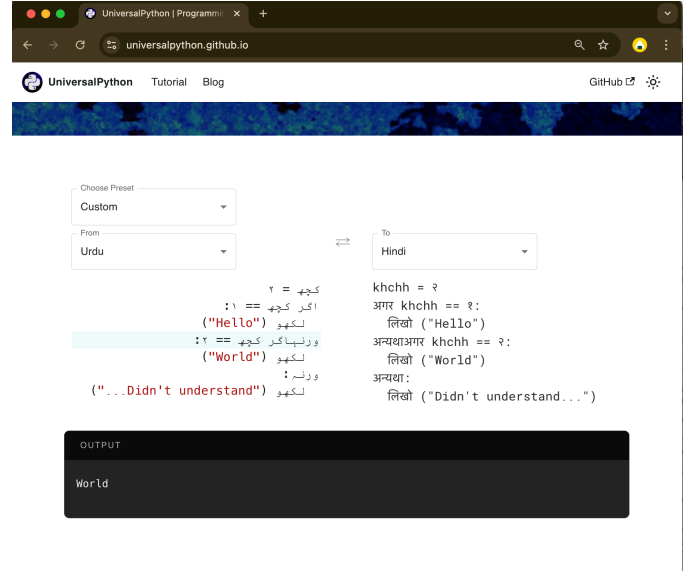


Figure 2: Urdu to Hindi code translation and successful compilation with UniversalPython.

and achieved a working kernel for UniversalPython. It works line-by-line while maintaining the program memory. This also gave us a visual interface for the language to test it thoroughly.

## VI. LIMITATIONS

- Third-party library support is vital for any software platform to flourish. While third-party libraries do not cause the program to crash, we would prefer a better developer experience in which third-party libraries are also translated.

- It is currently not possible to translate words with spaces in between them. For example, ‘elif’ in Python actually translates to *ورنہ اگر* (notice the space in the middle) in the Urdu language. Some languages may also have different placements for verbs and nouns, and different words for different quantities (e.g. mono, bi, tri). This may present a challenge for UniversalPython, where more creative effort may be needed to construct a better front-end experience for programmers, while ensuring compatibility with Python.

- Translation quality is very weak for certain low-resources and contextual languages.

## VII. FUTURE WORK

UniversalPython can be developed as an extensible programming language which would grow alongside Python and would be interoperable with it. It is also maintainable, because since it is simply a wrapper, only the keywords need to be updated in the dictionary, and it will work for any future (or past) versions of Python. Furthermore, it is scalable to other languages because the dictionary structure allows for any language to be added to UniversalPython.

Table III: Comparing UniversalPython with other non-English, monolingual programming languages and multilingual programming languages using "A Framework for the Localization of Programming Languages" [25]

Aspect/Prog Language	UniversalPython	Scratch	Snap!	Dolittle	Ebda3	Qalb	Wenyan	Excel	PSeInt	Rapture	H
Language	Multi	Multi	Multi	Japanese	Arabic	Arabic	Chinese	Multi	Spanish	Russian	M
Alignment	N	N	N	N	N	T	N	T	N	T	
Non-English keywords	•	•	•	•	•	•	•	•	•	•	
Non-Latin variable names	•	•	•	•	•	•	•	◦	•	•	
Non-English productions	•	◦	•	-	-	•	◦	•	-	-	
Non-English numerals	•	-	-	-	-	•	•	•	-	-	
Characters without meaning	•	-	-	-	-	•	•	-	-	-	
Diacritics	•	•	•	◦	◦	•	◦	-	•	◦	
Alternative keywords	•	-	-	-	◦	-	•	-	•	◦	
Localized punctuation	•	◦	◦	◦	◦	◦	•	•	-	-	
Right to left support	•	◦	-	◦	•	•	◦	•	-	-	
Multi-lingual programming	•	-	-	-	-	-	-	-	-	•	
Error messages	-	◦	◦	•	◦	◦	-	•	•	-	
Multi-lingual 3rd-party libraries	◦	◦	-	◦	◦	◦	◦	•	-	-	

To improve the current implementation, it would be useful to accommodate as many Python keywords as possible, preferably all of them. It would also be useful to separately have translations for major libraries/packages such as pandas, numpy, cv2, etc. A scalable and efficient method for this would be to automatically scan such libraries (the Python source code) to identify important keywords, automatically translate them to the nearest and best possible Urdu word, and add them to the Grammar. This can be achieved using Natural Language Processing, Deep Learning, Machine Learning and similar methods.

This framework could take on a similar role as TypeScript does in the JavaScript world, where it is usually used as a build-time transpiler to JavaScript [30]. To make it easy for third-party libraries to integrate with UniversalPython, inspiration can be taken from TypeScript's *\*.d.ts* files — in UniversalPython's case, it could be an *interfaces.<language\_code>.yaml* file containing the translations for the functions and variables, in a predefined format. Again, we leave this to the imagination of the reader.

In the current framework, making it possible for the user to *choose* whether or not to detect variables and functions, and translate them with machine translation into their corresponding names, could make the experience better.

In order to increase the accuracy of the translations, a translation-verification system can be developed, where beginner and experienced programmers can be asked, "How would you write the following snippet of code in your language?", and let them write the code in any shape or form they desire. This could be done with data from platforms like Leetcode, however, we leave that to the imagination of the reader.

It would also be helpful to conduct more User Experience tests with programmers across the globe, so that it can be determined whether or not UniversalPython really does benefit people who speak non-English languages to learn programming.

Designing and developing a better evaluation metric to test the effectiveness of UniversalPython and similar

multilingual programming languages / frameworks.

A REPL can be made by hooking into the Python REPL. Integrations can be made with Replit, Leetcode etcetera.

Support from official Python could boost the project a lot.

#### ACKNOWLEDGMENT

I would like to acknowledge my mentor, Dr Omer Beg, for always guiding me to the right path during my Bachelor's, professional life, and Master's degree.

#### REFERENCES

- [1] Y. Qian, P. Yan, and M. Zhou, "Using data to understand difficulties of learning to program: A study with chinese middle school students," in *Proceedings of the ACM Conference on Global Computing Education*, ser. CompEd '19. New York, NY, USA: Association for Computing Machinery, 2019, p. 185–191. [Online]. Available: <https://doi.org/10.1145/3300115.3309521>
- [2] Wikipedia contributors, "Non-english-based programming languages — wikipedia, the free encyclopedia," [Online; accessed 7-December-2024].
- [3] J. Backus, "The history of fortran i, ii, and iii," *ACM Sigplan Notices*, vol. 13, no. 8, pp. 165–180, 1978.
- [4] K. Zuse, "Ansätze einer theorie des allgemeinen rechnens unter besonderer berücksichtigung des aussagenkalküls und dessen anwendung auf relaisschaltungen," Deutsches Museum, Tech. Rep., 12 1963, typoskript. Vgl. NL 207/0211, NL 207/0219 u. NL 207/0222. [Online]. Available: <https://digital.deutsches-museum.de/item/NL-207-0281/>
- [5] F. L. Bauer and H. Wössner, "The "plankalkül" of konrad zuse: a forerunner of today's programming languages," *Communications of the ACM*, vol. 15, no. 7, pp. 678–685, 1972.
- [6] D. Bühmann, *Mother tongue matters: Local language as a key to effective learning*. UNESCO, 2008.
- [7] K. Taylor-Leech, "Finding space for non-dominant languages in education: Language policy and medium of instruction in timor-leste 2000–2012," *Language Planning for Medium of Instruction in Asia*, pp. 119–136, 2015.
- [8] B. Trudell, "Local-language literacy and sustainable development in africa," *International Journal of Educational Development*, vol. 29, no. 1, pp. 73–79, 2009.
- [9] U. Bangkok, "Advocacy brief: Mother tongue-based teaching and education for girls," 2005, p. 2.
- [10] H. Gedawy, S. Razak, and H. Alshikhabobakr, "The effectiveness of creating localized content for middle school computing curriculum," in *Proceedings of the 2019 ACM Conference on Innovation and Technology in Computer Science Education*, ser. ITiCSE '19. New York, NY, USA: Association for Computing Machinery, 2019, pp. 478–484. [Online]. Available: <https://doi.org/10.1145/3304221.3319778>

- [11] S. Dasgupta and B. M. Hill, "Learning to code in localized programming languages," in *Proceedings of the fourth (2017) ACM conference on learning@ scale*, 2017, pp. 33–39.
- [12] E. Steinfeld, *Universal Design: Creating Inclusive Environments*. John Wiley & Sons, Inc, 2012.
- [13] P. J. Guo, "Non-native english speakers learning computer programming: Barriers, desires, and design opportunities," in *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*, ser. CHI '18. New York, NY, USA: Association for Computing Machinery, 2018, pp. 1–14. [Online]. Available: <https://doi.org/10.1145/3173574.3173970>
- [14] . , " , " , pp. 72–85, 1982.
- [15] E. Olatunji, J. Oladosu, O. Odejebi, and S. Olabiyisi, "Design and implementation of an african native language-based programming language," *International Journal of Advances in Applied Sciences*, vol. 10, p. 171, 06 2021.
- [16] S. Kadam, "Kalaam - programming language in hindi." [Online]. Available: <https://github.com/Kalaam-Programming-Language/Kalaam>
- [17] L. Huang, "Wenyan-lang: A programming language for the ancient chinese," <https://github.com/wenyan-lang/wenyan>, 2019, accessed: 2024-12-08.
- [18] A. Community, "Alif arabic programming language," GitHub repository, 2024. [Online]. Available: <https://github.com/alifcommunity/Alif>
- [19] F. Hermans, "Hedy: A Gradual Language for Programming Education."
- [20] M. Resnick, J. Maloney, A. Monroy-Hernández, N. Rusk, E. Eastmond, K. Brennan, A. Millner, E. Rosenbaum, J. Silver, B. Silverman *et al.*, "Scratch: programming for all," *Communications of the ACM*, vol. 52, no. 11, pp. 60–67, 2009.
- [21] W. N. Freitas Filho, "Scratch software as a possibility for mathematics classes," Course Completion Work (Undergraduate Degree in Mathematics), Uberlândia, 08 2022, uRI: <https://repositorio.ufu.br/handle/123456789/36259>. [Online]. Available: <https://repositorio.ufu.br/handle/123456789/36259>
- [22] P. Wang, "Pseutopy: Towards a non-english natural programming language," in *Proceedings of the 17th ACM Conference on International Computing Education Research*, 2021, pp. 429–430.
- [23] A. Memon, "UrduScript," Aug. 2019. [Online]. Available: <https://www.github.com/asadm/urduScript>
- [24] —, "Urduscript: Urdu mein programming," <https://asadmemon.com/urduScript/>.
- [25] A. Swidan and F. Hermans, "A framework for the localization of programming languages," in *Proceedings of the 2023 ACM SIGPLAN International Symposium on SPLASH-E*, 2023, pp. 13–25.
- [26] N. Coghlan, "The transition to multilingual programming with python," Red Hat Blog, 9 2014. [Online]. Available: <https://developers.redhat.com/blog/2014/09/09/transition-to-multilingual-programming-python>
- [27] M. P. Edgar, Beomus, A. Buhai, A. Ramzaev, J. Barzilaj, Sergio, J. Yeung, A. Singh, s. taneja, R. P. Mishra, and atom19 i, "Legesher." [Online]. Available: <https://github.com/legesher/legesher>
- [28] J. Otten, A. Anastasopoulos, and K. Moran, "Towards a universal python: Translating the natural modality of python into other human languages," in *2023 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, 2023, pp. 352–357.
- [29] T. A. Community, "The algorithms - python," <https://github.com/TheAlgorithms/Python>.
- [30] G. Bierman, M. Abadi, and M. Torgersen, "Understanding typescript," in *ECOOP 2014 – Object-Oriented Programming*, R. Jones, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 2014, pp. 257–281.