Breaking the Programming Language Barrier: Multilingual Prompting to Empower Non-Native English Learners

James Prather
Abilene Christian University
Abilene, Texas, USA
james.prather@acu.edu

Brent N Reeves
Abilene Christian University
Abilene, Texas, USA
brent.reeves@acu.edu

Paul Denny
The University of Auckland
Auckland, New Zealand
paul@cs.auckland.ac.nz

Juho Leinonen
Aalto University
Espoo, Finland
juho.2.leinonen@aalto.fi

Stephen MacNeil
Temple University
Philadelphia, Pennsylvania, USA
stephen.macneil@temple.edu

Andrew Luxton-Reilly
The University of Auckland
Auckland, New Zealand
a.luxton-reilly@auckland.ac.nz

João Orvalho
Polytechnic University of Coimbra
Coimbra, Portugal
orvalho@esec.pt

Amin Alipour
University of Houston
Houston, Texas, USA
maalipou@central.uh.edu

Ali Alfageeh
University of Houston
Houston, Texas, USA
aalfagee@cougarnet.uh.edu

Thezyrie Amarouche
University of Toronto Scarborough
Toronto, Canada
thezyrie.amarouche@mail.utoronto.ca

Bailey Kimmel
Abilene Christian University
Abilene, Texas, USA
blk20c@acu.edu

Jared Wright
Abilene Christian University
Abilene, Texas, USA
jpw19b@acu.edu

Musa Blake
Abilene Christian University
Abilene, Texas, USA
mbb23c@acu.edu

Gweneth Barbre
Abilene Christian University
Abilene, Texas, USA
gab23c@acu.edu

Abstract

Non-native English speakers (NNES) face multiple barriers to learning programming. These barriers can be obvious, such as the fact that programming language syntax and instruction are often in English, or more subtle, such as being afraid to ask for help in a classroom full of native English speakers. However, these barriers are frustrating because many NNES students know more about programming than they can articulate in English. Advances in generative AI (GenAI) have the potential to break down these barriers because state of the art models can support interactions in multiple languages. Moreover, recent work has shown that GenAI can be highly accurate at code generation and explanation. In this paper, we provide the first exploration of NNES students prompting in their native languages (Arabic, Chinese, and Portuguese) to generate code to solve programming problems. Our results show that students are able to successfully use their native language to solve programming problems, but not without some difficulty specifying programming terminology and concepts. We discuss the challenges they faced, the implications for practice in the short term, and how this might transform computing education globally in the long term.

CCS Concepts

• **Human-centered computing** → **Empirical studies in HCI**; **User studies**; **Natural language interfaces**; **User interface programming**; • **Computing methodologies** → **Artificial intelligence**; • **Social and professional topics** → **Computing education**; **Computer science education**; **CS1**.

Keywords

AI; Artificial Intelligence; Automatic Code Generation; Codex; Copilot; CS1; GenAI; GitHub; GPT; GPT-4; ChatGPT; HCI; Introductory Programming; Large Language Models; LLM; Non-Native English Speakers; Novice Programming; OpenAI; Prompt Problems

ACM Reference Format:
James Prather, Brent N Reeves, Paul Denny, Juho Leinonen, Stephen MacNeil, Andrew Luxton-Reilly, João Orvalho, Amin Alipour, Ali Alfageeh, Thezyrie Amarouche, Bailey Kimmel, Jared Wright, Musa Blake, and Gweneth Barbre. 2025. Breaking the Programming Language Barrier: Multilingual Prompting to Empower Non-Native English Learners. In *27th Australasian Computing Education Conference (ACE 2025), February 12–13, 2025, Brisbane, QLD, Australia.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3716640.3716649

ACE 2025, Brisbane, QLD, Australia

ACM ISBN 979-8-4007-1425-2/25/02
https://doi.org/10.1145/3716640.3716649

1 Introduction

Non-native English speakers (NNES) face significant challenges when learning programming through English language instruction. They tend to set higher academic goals for themselves and spend more time studying compared to their native English-speaking peers [23]. The stance that one must know English to be a programmer is quite entrenched in the broader community[1], but this view is often critiqued [24, 46]. The impacts of learning English-adjacent programming languages through English language instruction are not well understood. Although NNES sense of belonging does not appear to be impacted by learning programming in English, other underlying factors can lead to a less inclusive environment. On the one hand, these students often express higher self-doubt and embarrassment by asking for help [3, 47]. On the other hand, students enjoy learning programming in their native language and report that it positively impacts their experience [52]. However, instruction in native language does not seem to impact learning outcomes for NNES students [3, 59]. While one might think that English language instruction would vary across the globe being sensitive to local contexts, a recent study showed that it was remarkably monolithic across three different continents [5].

In 2019, Becker wrote that native natural language programming would have obvious advantages for NNES but remained a far-off fantasy [4]. The advent of generative AI (GenAI) since then has radically shifted our perception of the role that natural language might play in programming [14, 49]. Initial work on generating programming exercises using GenAI found that it could create coherent and customizable programming assignments [55]. Follow-up work showed that students in Finland preferred customized programming assignments, albeit in English [42]. Recent work has demonstrated the capabilities of ChatGPT-3.5 to generate programming problems not only in English but also in Tamil, Spanish, and Vietnamese [31]. Other types of programming problems, such as the classic "Explain in Plain English" (EiPE), are also being automated by GenAI that can be used to grade them at scale [15, 33]. These advancements offer a unique opportunity to explore expressing programming concepts in a learner's native language.

In this paper, we present the first exploration of NNES prompting GenAI to solve programming problems using their native language. We do this by utilizing Prompt Problems with prompts in a learner's native language. Prompt Problems are a type of programming exercise designed to teach programming concepts via GenAI prompting. Students receive a problem visually and must write a prompt that can generate code to solve the problem. If the generated code does not pass the suite of test cases, then they must edit their prompt again, iterating until it successfully passes. Initial work has shown that students find Prompt Problems engaging and encourage metacognitive reflection [13]. Here, we combine several threads of recent work to see if students can solve these problems in their native languages (Chinese, Portuguese, Arabic) and how it impacts their learning experience.

Therefore, our research questions are:

(1) **RQ1:** How successful are students at solving Prompt Problems in their non-English native languages?
(2) **RQ2:** How does solving Prompt Problems in non-English native language impact user experience?

2 Related Work

2.1 LLMs in Computing Education

Since the popularization of large language models (LLMs) with the release of ChatGPT in November 2022, researchers in computing education have explored both the opportunities and challenges presented by these models.

2.1.1 Opportunities. One of the earliest identified opportunities of LLMs for computing education is the automatic generation of programming exercises. Sarsa et al. studied the capabilities of Codex in generating novel programming exercises focusing on specific themes (such as 'cooking' or 'basketball') and concepts (such as 'for-loops') [55]. Their findings suggested that LLMs can create novel exercises that are often good enough to provide to students as-is. Follow-up work has found that while LLMs are good at generating high-quality exercises [11, 42], the thematic contextualization of created content is often shallow [42].

In addition to programming exercises, LLMs have been used to create multiple-choice questions (MCQs) for computing courses [1, 17, 62]. Tran et al. found promising results when using GPT-4 to generate multiple-choice questions that were isomorphic to provided examples [62]. In similar work, Doughty et al. examined GPT-4's performance for generating MCQs that would align with course learning objectives [17]. Their results suggest that GPT-4 was able to generate very high-quality MCQs that were evaluated to be of similar quality as those generated by human educators.

Researchers have also explored using LLMs to help students understand code. Recent work by Bernstein et al. had students create personally meaningful analogies for recursion using ChatGPT, a notoriously difficult threshold concept [6]. The analogies created by students with the help of ChatGPT were more diverse compared to generic analogies that LLMs would generate on their own. Additionally, the participants reported that these analogies helped them understand recursion.

LLMs have also been used to explain both program code [32, 37, 44, 55] and programming error messages [38, 54, 61, 65]. Sarsa et al. found that Codex was able to explain program code in natural language correctly for approximately two thirds of the lines of code in the examples they provided it [55]. MacNeil et al. reported that students found LLM-generated code explanations useful and helpful for their learning [44]. Leinonen et al. found that code explanations generated by LLMs were rated as being more accurate and easier to understand compared to student-generated explanations [37]. For programming error messages, while Codex was only sometimes helpful in enhancing them [38], follow-up work using GPT-3.5 and GPT-4 has found improved results. Taylor et al. found that GPT-3.5 explained errors correctly in up to 90% of cases [61], and Wang et al. found that students who were provided enhanced programming error messages created by GPT-4 repeated errors less frequently and resolved errors with fewer attempts [65]. However, Santos et

[1]Reinforced by documents such as PEP 8 – a style guide for Python – with guidance such as *Python coders from non-English speaking countries: please write your comments in English.*

al. found that the time to fix errors was not improved by providing GPT-4 enhanced error messages [54].

LLMs have also been explored for providing direct assistance to students, either through responses to help requests [25] or by feedback on program code [34, 35]. Hellas et al. explored how well LLMs could respond to students' help requests and found that while GPT-3.5 would often detect issues in student's incorrect code, it would also hallucinate nonexistent issues [25]. Kiesler et al. found that ChatGPT was able to detect and correct compiler errors well, but had lower performance for other types of errors [34]. Koutcheme et al. used multiple smaller, open-source LLMs and found that some of the open-source models such as Zephyr-7B-beta rivaled some proprietary models such as GPT-3.5 in performance [35].

2.1.2 Challenges. Despite the many opportunities that LLMs provide, there are challenges too. The earliest computing education work that utilized LLMs found that they can solve most introductory programming (CS1) exercises [19], and follow-up work with more recent models has demonstrated rapidly improving performance [49]. Their problem solving is not limited to programming exercises, as LLMs have been found to solve Parsons problems too [53], even if provided as images [27], as well as computer graphics questions requiring both visual perception and geometric reasoning skills [18]. LLMs can also solve multiple-choice questions related to programming [56]. This has raised fears that students might use LLM support for academic misconduct, or over-rely on LLM support even when their intention is not to cheat. While some work has looked into automatically detecting LLM-generated solutions [26, 48], there are no sure-fire methods to conclusively classify code as LLM-generated.

Perhaps more concerning is that students could inadvertently be negatively affected by LLM use. Prather et al. replicated an earlier study [50] looking into metacognitive difficulties that novice programmers face while they're programming, but this time giving students access to LLM tools such as GitHub Copilot and ChatGPT [51]. They found that not only did students still face the same metacognitive difficulties, but some of these were exacerbated by the LLMs and new difficulties were also introduced. While the best students were able to accelerate with the help of GenAI tools, students who were already struggling faced more difficulties, which could widen the gap between the best and the worst performing students. This result is similar to help-seeking, where Hou et al. found that students who are capable at using models get the most benefits [28]. These issues highlight the need for novel pedagogies that help students to learn to successfully leverage GenAI models in their work.

2.2 Emerging LLM-Powered Pedagogies

Teaching students to write natural language prompts is an emerging area in computing education, and can be applied equally well to support both code writing and code comprehension tasks. Moreover, given the language translation capabilities of LLMs, pedagogies based on this idea could be applicable in any spoken language, potentially improving the accessibility of programming education for diverse student populations.

2.2.1 Prompt Problems. When exploring the performance of LLMs for solving typical CS1 problems, Denny et al. observed that making certain refinements to the prompts led to greater accuracy in the generated code [12]. They argued that learning how to craft effective prompts is essential for novice programmers, but did not directly propose teaching strategies for developing this skill. In subsequent work, Denny et al. introduced the idea of 'Prompt Problems' as a novel exercise for helping students learn how to solve computational tasks through natural language prompts [13]. In a Prompt Problem, a student would be shown a computational task, typically presented visually without any textual description, and they would write a natural language prompt for an LLM to generate code to solve that task. Prompt Problems allow shifting the focus from syntax mastery, often emphasized early in programming education, towards higher-level problem-solving. The authors presented and evaluated a tool called Promptly which would execute the generated code against predefined test cases to assess both the effectiveness of the prompt and the student's understanding of the problem. Feedback on the activity from students in both a CS1 and CS2 course suggested that they appreciated that it engaged their computational thinking skills and introduced them to new programming constructs. However, it also highlighted the need for further research into how these exercises could be best integrated into classroom practice and whether they could meaningfully improve learning outcomes. In addition, all prior research involving Prompt Problems has been in the context of English-language prompting.

2.2.2 Explain in Plain Language (EiPL). While Prompt Problems can be viewed as an alternative approach to solving *code writing* tasks, by evaluating code generated from a student's prompt against a test suite, an analogous approach can be used to provide feedback on *code comprehension* tasks. There is a considerable body of research in the computing education literature on 'Explain in Plain English' (EiPE) questions, which require students to articulate the purpose of a code fragment in natural language. Early work exploring this type of question by Whalley et al. identified a connection between students' programming skills and their ability to describe code accurately in simple terms [66]. Over the following decade, further research validated this approach across various contexts [40, 43, 64]. In 2014, Corney et al. highlighted that EiPE tasks enhance students' ability to reason about code, which can, in turn, improve their coding skills [9], however scaling these exercises has been challenging due to the subjective nature of grading free-text responses.

Applying LLMs to the task of grading EiPE responses has shown great promise [15]. Smith et al. proposed 'Code Generation Based Grading' (CGBG) [58], a method conceptually similar to the idea for generating feedback on Prompt Problems, by using LLMs to generate code based on a student's EiPE response. The generated code is then tested against predefined test cases, offering both objective grading and actionable feedback for students. Their study demonstrated that CGBG aligns well with human grading while providing a scalable solution for large classes, making it a promising approach for automated assessments. Smith et al. further explored the connection between prompt writing and code comprehension [57]. Their research showed that students benefit from the dual challenge of writing prompts and understanding generated code. By

performing both tasks, students not only develop their code comprehension skills but also gain proficiency in prompt engineering. Similar recent work has shown that LLM-grading of EiPE questions is engaging for students, but importantly also found that relational responses, where students integrated code elements into high-level summaries, were the most successful in generating correct code [15]. This line of research highlights the potential for LLM-powered exercises to foster higher-order thinking, and to help students appreciate the connection between natural language prompts and the code they generate. Very recently, Kerslake et al. examined the integration of both Prompt Problems and EiPE questions into a large introductory programming course [33]. A key finding of this work was that students who struggled with traditional coding tasks performed better with natural language prompts, suggesting that these tasks engage a broader range of cognitive skills and provide a more accessible entry point for novice programmers.

2.2.3 Multilingual Support in Programming Education. Non-native English-speaking students often face significant challenges in programming education [3, 22, 23]. For example, Becker highlighted how the predominance of English in programming languages, documentation, and error messages adds a significant cognitive load to non-native speakers, making it harder for them to fully participate in programming courses [4]. This study revealed that non-native speakers often encounter difficulties interpreting keywords and comments, increasing their cognitive load and limiting their ability to focus on problem-solving. Although instructors have been shown to adjust their speech patterns and vocabulary to meet the needs of diverse student groups [5], modern LLMs have suddenly opened the door to providing significant multilingual support in computing education. Recent work by Jordan et al. explored the use of LLMs to generate programming exercises in Spanish, Vietnamese, and Tamil [31]. Although the model they used at the time of their research is no longer state of the art, support for generating resources in Spanish and Vietnamese was generally good. Despite some limitations, which will likely lessen over time as model capabilities improve, the authors see great potential in using LLMs as a pathway for creating culturally relevant programming resources tailored to non-native speakers. This is a particularly important result given that when students see their identities reflected in learning activities, they experience positive academic and social outcomes, making computer science more relevant and accessible to them [30].

There has recently been some early exploration of multilingual support for prompting-based activities in computing education. Smith et al. explored the use of EiPE questions in Indic languages such as Hindi, Tamil, and Marathi, revealing both opportunities and challenges in supporting multilingual learners [29]. While students appreciated the ability to engage with programming tasks in their native languages, many still preferred English for technical precision and familiarity. This suggests that while multilingual support can lower accessibility barriers, students' existing preferences and technical contexts must also be considered when designing programming exercises.

The relationship between programming and natural languages has been further explored by Veldthuis and Hermans who applied natural language vocabulary acquisition models to programming education [63]. Their study restructured introductory programming lessons to include strategies typically used in second language learning, such as scaffolding vocabulary – the gradual introduction of new concepts in phases, with opportunities for practice and verification. Using the Hedy programming environment, students were progressively exposed to increasing complexity, initially permitting errors in syntax to mirror natural language learning processes. The researchers found that these strategies not only enhanced students' understanding of programming concepts but also improved engagement and motivation. This approach aligns with the broader goal of our present study, which investigates how LLMs can support multilingual Prompt Problems by offering adaptive, language-aware programming exercises that accommodate students' linguistic backgrounds and lower cognitive barriers to learning.

3 Methodology

We explored the use of Prompt Problems in three distinct environments: a university with English as the language of instruction, but where NNES students completed tasks using Chinese; a Portugese university where students completed tasks using Portugese; and, a Middle-Eastern university in which students used Arabic. In each institution, the prompts used by students were collected along with a post-survey.

3.1 Data Collection

The first data collection was conducted in February 2024 at Polytechnic University of Coimbra in Portugal with 27 students whose native language was Portuguese. Students were in a CS2 course learning Python and instructed in Portuguese. The second data collection was conducted in May 2024 at the University of Auckland, New Zealand, in a post-graduate course with 19 students using Python. The language of instruction was English, but most of the students enrolled in the course were non-native English speakers whose primary language was Chinese. The third data collection was also in May 2024 at Umm Al-Qura University in Saudi Arabia with 34 students. These students were undergraduates learning in Java and the language of instruction was Arabic. In each context, the instructor explained the concept of Prompt Problems beforehand and it was demonstrated to them. Students were then invited to participate by completing the Prompt Problems shown in Table 1 by prompting in their native (i.e. non-English) language. After the activity, a short post-survey was administered to capture their experience solving Prompt Problems in their native language.

3.2 Analysis

We performed a quantitative analysis of the prompt submission data (RQ1) and a qualitative analysis of the survey data (RQ2).

3.2.1 Quantitative Analysis. The initial dataset included 1,771 total prompts by students in all three groups. Because we were only interested in the prompts by NNES students, we cleaned the data by removing all prompts by students who only prompted in English. While it's possible that a NNES student decided to only prompt in English, our research questions are directed at the experience of students prompting in their native language. Therefore, all of the 1679 remaining prompts contained at least some non-English words. We then calculated completion data for each problem for each language group.

Table 1: Prompt Problems used in the three studies

Name	Description	Example
1. scramble	Write a Python function called `scramble` that accepts a string containing only lowercase letters and a number as a parameter. It returns a string with each letter shifted by the number.	`scramble("zoo", 2) => 'bqq'`
2. arrange	Write a function in Python called `arrange` that accepts a string and returns a new string with the letters sorted: capital letters in ascending order followed by lowercase letters in descending order.	`arrange("AaBbCcDd") => 'ABCDdcba'`
3. speak	Write a Python function called `speak` that replaces characters with numbers: 'e' -> '3', 'o' -> '0', 's' -> '5', 't' -> '7', 'a' -> '4', 'i' -> '1', including their uppercase equivalents.	`speak("Hello World!") => 'H3ll0 W0rld!'`

After visually examining the text of the remaining prompts, two researchers created six categories of prompts based on the amount of English, native language, syntax, or code in the prompt. Four other researchers then categorized every prompt into one of those categories. If there was concern about which category to place a prompt into, the four researchers categorizing discussed the prompt and came to consensus. These decisions were randomly sampled for quality assurance purposes by the first two researchers who made the categories. The groups were as follows:

- **O:** Other (empty prompt, URL, etc.)
- **E:** English, except for a single non-English word or very short phrase
- **N:** Non-English language except for computer language terms / jargon (examples include: string, char, array, function, def, function names, result, for, repeat, in, text, change, package, import, etc.)
- **M:** Mixed
- **C:** All code

These categories represent the kinds of linguistic prompting strategies that students used to solve the Prompt Problems. We analyzed these data to find the most common strategies used by different language groups as well as the kinds of state changes that students went through while solving the problem.

3.2.2 Qualitative Analysis. The data was coded by a member of the research team following a reflexive thematic analysis as outlined by Braun and Clarke [7]. This approach highlights the researcher's active role in the development of themes. This inductive approach is particularly suited to exploratory research with smaller, diverse samples. Given the relatively small sample size and the heterogeneity between the three participant groups, an inductive strategy facilitated the generation of themes grounded in the data, rather than being constrained by preconceived theoretical frameworks.

The primary goal of the analysis was therefore to identify patterns and trends that can contextualize our quantitative findings and to inform new hypotheses. Unlike in positivist methods, where sample size is predetermined based on statistical power, thematic analysis does not require a set number of participants. Instead, our process was guided by thematic saturation, which is a point at which new responses cease to produce new insights [21]. In our analysis, thematic saturation was considered with regards to each participant linguistic group. Each new sample provided additional insights, but within groups, saturation was often observed within the first 15 responses. Responses consistently aligned with the primary themes that emerged early in the coding process for each sample, and despite minor variations in individual responses, no new dominant themes were identified during the latter stages of analysis. A limitation that arises from this analysis is that our findings are suggestive, but not conclusive or capable of generalizing beyond the participants sampled in this study.

Table 2: Summary of student success rate by language and question

Language	Problem 1	Problem 2	Problem 3
	Students Attempting / Correct		
Arabic	42 / 27	26 / 15	14 / 12
Chinese	12 / 12	12 / 5	4 / 2
Portuguese	22 / 21	21 / 20	19 / 18

Table 3: The total number of prompts that used a particular strategy with the number of correct prompts that used that strategy in parenthesis.

Language	Linguistic Prompting Strategy Counts				
	Other	English	Native	Mixed	Code
Arabic	4 (0)	7 (0)	1123 (55)	18 (2)	0 (0)
Chinese	0 (0)	26 (6)	40 (6)	22 (5)	4 (2)
Portuguese	0 (0)	4 (2)	428 (76)	2 (2)	0 (0)

4 Results

4.1 Quantitative Data

The data presented in Table 2 show that a majority of students were able to complete the exercises using at least some of their native language. As noted in previous work on Prompt Problems, the sequential nature of the tasks means there are naturally fewer students who attempt subsequent problems [13].

The data from the categorization (see Table 3) show that most students attempted to prompt in their native language (category "N"). The next highest categories were "M" (Mixed language) and "E" (almost all English except for one native language word or phrase). Given that students were able to solve the Prompt Problems (see Table 2) and that most prompted in their native language almost

exclusively, these data show that students were able to successfully solve the Prompt Problems in their native language. However, there is a clear divide between correct solutions in Chinese and Portuguese compared to Arabic. For category "N", Chinese had a 15% success rate with that prompting strategy and Portuguese had an 18% success rate. Prompts that were in Arabic, however, had a 5% success rate. Interestingly, students prompting in Chinese resorted to using English the most and found some success with it.

The vast majority of submission streams (prompt submissions attempting to answer one of the Prompt Problems) used one category exclusively. We had expected to find many more submission streams that included more than one state. Of 172 successful submission streams, 152 used only 1 state, and 127 of those were *Native*. Nine streams used two states, meaning that, for example, after submitting a prompt that was categorized as *Native*, the student switched to, for example, *English*. Of those streams that had two states, most were *English*-then-*Native* or *Native*-then-*English*. Nine used three, one used five, and one used six states. *Native* was the most frequent starting state as well as the most frequent ending state.

4.2 Qualitative Data

Across the survey responses, three primary themes were identified regarding the use of native versus foreign language. First, participants shared that models generally performed poorly when interacting in their native language. This poor performance appeared to be more pronounced for the Arabic-speaking sample, a finding which can be seen in the quantitative data above. Second, participants highlighted trade-offs between using their native language and communicating in English, often finding English leads to better results, despite being less expressive for them. Finally, some participants mentioned that while their native spoken language was not English, they were more accustomed to coding in English, suggesting it felt more natural to use English in the context of programming. In the following subsections, participants are labeled by language: PA = Participant Arabic, PC = Participant Chinese, PP = Participant Portuguese.

4.2.1 Poor Support for Some Languages. A recurring theme across all participant samples was the inadequate support LLMs provide for some languages compared to others. This observation is consistent with prior research in natural language processing, which refers to these languages as 'low-resource'—languages that have less data available in the original model's training set [45]. For example, PA-1 explained how ChatGPT can face challenges when understanding the Arabic language:

> *"There are challenges with ChatGPT's understanding of the Arabic language."* (PA-1)

Anecdotally, this theme was much more prevalent in responses in the Arabic-speaking sample than in the Chinese-speaking sample. This again relates to the concept of 'low-resource' languages as there is currently more Chinese text included in training sets for large language models.

One participant speculated that this might stem from code often being written in English, which creates a gap when using other languages to interact with code or to explain technical concepts.

> *"In English, it's easy to write what you want because the programming is in English, while in Arabic, it was a bit difficult to convey the information to ChatGPT"* (PA-4)

Due to this poor performance for native languages, multiple participants expressed an explicit preference for using English rather than their native language when interacting with language models.

> *"It was somewhat bad, but I think it would be much better if it were in English."* (PA-14)

Interestingly, some participants implied that using their second language, English, had an unexpected benefit. They described how writing prompts in English required them to be more deliberate and considerate about the language they used, which forced them to slow down and organize their thoughts more carefully.

> *"I'm so familiar with native language, so the advantage is I can write the command more smoothly, while this makes my command omit some points, which makes the model hard to understand. [To] use English, I need to organize my language and always consider about the grammar, so it might be a little bit slow, but accurate."* (PC-16)

This observation suggests that slowing down and planning may offer metacognitive benefits, as planning is an important metacognitive strategy. However, more research is needed to systematically investigate this potential effect.

Despite the poor performance claimed by many participants, a small minority of participants expressed a preference for their native language. For example PA-24 explained that Arabic was easier but required more effort for them to explain the requirements to the model:

> *"The Arabic language is easier, but it requires a lot of explanation."* (PA-24)

4.2.2 Trade-offs between Expressivity and Model Performance. Participants across the samples frequently shared about the trade-offs they face being more expressive in their native language and achieving better model performance when using English. Many participants used the word 'expressive' to describe how much easier it was to communicate their intent and goals in their native language. For example, PC-12 explained:

> *"Native language is more easy for me to express my thoughts. But more difficult for ChatGPT to understand me."* (PC-12)

And this experience was also shared by PA-4 in the Arabic-speaking sample:

> *"The advantages are that Arabic is my language, so I will have strong expression in it. However, the disadvantage I faced is that the program did not understand the explanation well."* (PA-4)

This trade-off was not unique to one language. For instance, PA-33 and others shared similar experiences about how language models did not appear to understand some of the words from their native language:

> *"Since it is my native language, it was much easier than English. [However,] it didn't understand some Arabic words."* (PA-33)

Another participant noted the balance between providing more detailed descriptions in their native language and achieving greater accuracy in English:

> *"I can give a more detail description with my native language, but seems english description can be more accurate"* (PA-2)

PC-15 attributed some of this trade-off to differences in meaning between the two languages:

> *"When I use my native language, I can express my ideas more accurately and avoid grammar problems to some extent. However, some of the same words may have very different meanings in my native language and in English, and also, many of the inflectional structures are completely different, which causes some problems."* (PC-15)

One participant speculated that the complexity of their descriptions in their native languages might have contributed to the model misunderstanding them. For example, PA-1 said:

> *"The abundance of vocabulary made it difficult for it to understand"* (PA-1)

There appears to be a trade-off between the ease and clarity of expressing one's thoughts in their native language and the performance benefits gained by code-switching to English. While participants found it more natural to articulate ideas in their own language, they believed using English provided better comprehension and accuracy. However, this trend was not seen among responses by Portuguese students. Nine students used the word "easy" and 13 used the word "positive" in their responses. One example is PP-15:

> *"Solving the exercises with Promptly was easy, I didn't have any difficulties and I found it quite intuitive."* (PP-15)

This may be because some languages, like European ones, would be considered "high-resource" languages and would therefore not suffer the same expressivity trade-offs as languages with non-Latin alphabets. The quantitative data correlates to this trend in the qualitative data where Portuguese had the highest rate of solving each problem (see Table 2) and also the highest rate of solving it in native language (see Table 3).

4.2.3 Writing Native English Code. While the participants are not native English speakers, many of them described how they had a lot of experience writing code in English and that trying to think about writing code in their own language presented a unique challenge. For example, PC-6 highlights how variable names did not translate well from Chinese to English:

> *"I think the advantage is using my native language, I can describe the situation better and the disadvantage may be the variable name is English and it can't be translated accurately."* (PC-6)

Similarly, PC-9 claimed that *"Certain words don't exist"*, which suggests that even when writing in their native language, participants had to incorporate English terminology. PC-13 echoed this sentiment, explaining that naming conventions in code still had to conform to English standards, regardless of the language used for other parts of the input:

> *"It behaves quite well in both English and Mandarin. but it have to make me explicitly name the function name so that it can pass."* (PC-13)

Several participants noted how deeply embedded the English language is in the context of programming, making it feel more natural to solve coding problems in English. This experience was captured by PC-3 as they said that trying to solve coding problems in their native language felt more like translation than direct problem-solving:

> *"I am really used to solve programming problems in english, answers, documentation, tutorials, youtube videos are mostly in English, trying to write them in my own language felt like I was translating my own thoughts. However writing prompts helps me have a look at the overall objective of a task and try to be precise which can be frustrating at first, but I guess it is useful in the long run."* (PC-3)

PA-3 reiterated this point simply saying that programming is in English and therefore it is easier to write what you want in English.

> *"In English, it's easy to write what you want because the programming is in English, while in Arabic, it was a bit difficult to convey the information to ChatGPT"* (PA-3)

5 Discussion

In this work, we investigated students' success when using their non-English native languages to solve Prompt Problems (RQ1) and examined how using native-language prompts influenced their experience (RQ2). While it is now widely known that ChatGPT and other LLMs are highly effective at solving programming problems [49] – particularly at the introductory level [19, 20] – their effectiveness with non-English prompts remains underexplored. As GenAI tools become increasingly integrated into computing education [14], understanding how they perform for non-English speakers is essential to fostering an inclusive learning environment.

Our findings indicate varying levels of success across language groups: students using Portuguese and Chinese achieved relatively high success rates, whereas Arabic speakers faced greater challenges in generating correct solutions. In terms of perceptions, although many students felt more able to express themselves using their native language, they often thought that using English was better for solving the Prompt Problems citing better performance and closer alignment with programming constructs. Several possible explanations may account for the challenges faced by students in this study.

5.1 Challenges of Non-English Prompting

Firstly, although state-of-the-art LLMs have been shown to have multilingual capabilities, they are most capable in English due to the simple fact that the vast majority of the data they have been trained on is in English [41]. In fact, although information on how proprietary models work internally is difficult to find, it appears

that most generative AI models will first translate a non-English prompt into English before attempting to answer it [60]. As a result of this English-centric bias, it is not surprising that models may be less effective at determining the intent of non-English prompts [16, 67]. Our results align with prior research that shows LLMs perform better with high-resource languages where more training data is available. Specifically, Arabic language data makes up less than 1% of the language distribution available in Common Crawl[2] (an open repository of web crawl data that makes up a significant portion of the training data for many LLMs), whereas Chinese is one of the most common languages in the corpus (although still much less frequent than English). This model bias may help to explain the particular difficulty faced by the Arabic speakers in our study.

Secondly, the output generated by the model is in a programming language that relies heavily on English syntax, which reduces the number of transformative steps needed when the input prompt is also in English. This alignment between the input (prompt) and output (program) language may also partially explain why students found writing prompts in English more effective. Related to this point, programming languages inherently encode concepts naturally expressed in English, for example, constructs like conditional branching using keywords such as `if` and `while`. These programming language keywords carry cultural and linguistic associations that may not translate directly into other languages [4].

Finally, the widespread use of English in programming has led students to become accustomed to writing code in English, even when it is not their native language. While students reported that it was easier to describe their thoughts in their native languages, they still found themselves relying on English technical terms, making the shift back to their native language feel cumbersome and unnatural. Thus, rather than simplifying the process, LLMs may be introducing an additional burden: students now have to translate their thoughts and intentions, as well as the syntax and technical terms. The first type of translation – converting their thought process into code – highlights how deeply embedded English is within programming, to the point where some participants are essentially thinking in a 'coding language' closely tied to English.

5.2 Implications for Practice

Despite the challenges we observed and have hypothesized, our findings also highlight the potential for using the language translation capabilities of LLMs to make Prompt Problems – and other kinds of new pedagogical assessments using GenAI – more broadly and globally accessible. This enables students to interact with programming concepts in their native languages, lowering barriers to entry and providing immediate access to programming activities and problem-solving tasks without being constrained by their English language abilities.

Our work contributes to the growing body of literature on broadening access to programming education for students with limited English proficiency. Kumar's *Refute* questions, for example, offer an alternative to Explain in Plain English (EiPE) questions by requiring students to identify incorrect logic rather than having to articulate explanations in English [2, 36]. Recent work by Smith et al. demonstrates that Code Generation Based Grading can allow students to answer code comprehension questions in multiple languages, including with high correctness rates across several Indic languages [29]. Similarly, we have observed that Prompt Problems offer students from diverse linguistic backgrounds a way to engage meaningfully with programming concepts in their preferred languages, promoting a more inclusive learning environment. While our data suggests that students were often frustrated by how intertwined English is with programming itself, our data also show that the radical leap in the abilities of GenAI over the last few years could mean students who grow up with it do not feel the same tensions. It's possible that the student of the future does not think programmatically in English because of these advances.

There is growing momentum in the field toward making computing education more culturally relevant by contextualizing the problems that are solved in ways that are relevant to the local community [8, 10, 39]. However, the Prompt Problems used in this study were largely abstract and lacked deep cultural relevance — it would be interesting in future work to explore the effectiveness of Prompt Problems in English and non-English environments where the problem is more closely aligned to the non-English culture and where there may be more natural ways to describe the problem in the non-English language. Designing problems that reflect culturally specific concepts and idioms may make non-English prompts more intuitive and accessible, enhancing students' learning experiences.

5.3 Limitations

This study has several limitations that could be addressed in future work. First, the three groups of learners differed in their educational backgrounds: the students prompting in Chinese were postgraduates, while those prompting in Portuguese and Arabic were undergraduates. This variation may have influenced the results, as postgraduate students are typically more experienced with programming concepts. Second, the programming languages of instruction were not the same across the groups. Students prompting in Arabic generated code in Java, while those prompting in Chinese and Portuguese generated code in Python. While these differences in programming language could influence performance, we believe the findings remain valid, as current AI models are proficient at generating correct code in multiple popular languages, including Java and Python. Moreover, this reflects an interesting strength of the study, as the results suggest that non-English prompting can work across different programming languages.

Third, the study focused on three languages – Arabic, Chinese, and Portuguese – which limits the generalizability of our findings to other languages. Variations in the linguistic structure and the representation of languages in the training data of AI models could lead to different outcomes for students whose native languages were not included in this study. Further research is needed to explore the effectiveness of native-language prompting in a broader range of languages, including those classified as low-resource languages.

Finally, the sample sizes for each language group were relatively small, and additional factors, such as individual language fluency, prior programming experience, and familiarity with AI tools, could influence students' success. These factors should be further investigated in future work.

[2]https://commoncrawl.github.io/cc-crawl-statistics/plots/languages

6 Conclusion

English has long been the primary language of programming instruction, with the keywords and syntax of most popular programming languages being English-centric. This has created a widely acknowledged barrier for non-native English speakers who must navigate both linguistic and technical challenges in computing courses. With the rise of generative AI (GenAI), and in particular the language translation capabilities of modern large language models, there is potential to lower this barrier by allowing students to approach problem-solving in their native languages. To investigate this potential, we introduced learners fluent in Chinese, Arabic, and Portuguese to Prompt Problems, a new kind of task in which programming exercises are solved by crafting natural language prompts instead of writing code directly. We found that students working in Portuguese and Chinese were generally able to achieve high success rates on these tasks, often expressing that native-language prompting allowed for a more natural articulation of their intent. In contrast, Arabic-speaking students appeared to face greater challenges, with lower accuracy and higher model misinterpretations. This finding may reflect the limited quantity of training data in Arabic that modern GenAI models have been trained on. Across all language groups, we observed a balance between expressivity and model performance – students often found it easier to formulate ideas in their native language, but English often yielded more accurate responses, especially for programming terms embedded in the English-language syntax. While we expect GenAI capabilities to continue to improve over time, especially for currently under-represented languages, our findings illustrate the transformative potential of existing multilingual GenAI tools. They can serve as valuable aids in democratizing programming education, making it more accessible and engaging for diverse learners globally.

Acknowledgments

This research was supported by the Research Council of Finland (Academy Research Fellow grant number 356114).

References

[1] Arav Agarwal, Karthik Mittal, Aidan Doyle, Pragnya Sridhar, Zipiao Wan, Jacob Arthur Doughty, Jaromir Savelka, and Majd Sakr. 2024. Understanding the Role of Temperature in Diverse Question Generation by GPT-4. In *Proc. of the 55th ACM Technical Symposium on Computer Science Education V. 2.* 1550–1551.

[2] Nimisha Agarwal, Viraj Kumar, Arun Raman, and Amey Karkare. 2023. A Bug's New Life: Creating Refute Questions from Filtered CS1 Student Code Snapshots. In *Proc. of the ACM Conf. on Global Computing Education Vol 1.* ACM, NY, NY, USA, 7–14.

[3] Vardhan Agarwal, Yada Chuengsatiansup, Elise Kim, Yuzi LYu, and Adalbert Gerald Soosai Raj. 2022. An Analysis of Stress and Sense of Belonging Among Native and Non-native English Speakers Learning Computer Science. In *Proc. of the 53rd ACM Technical Symposium on Computer Science Education - Volume 1.* ACM, NY, NY, USA, 376–382.

[4] Brett A. Becker. 2019. Parlez-vous Java? Bonjour La Monde != Hello World: Barriers to Programming Language Acquisition for Non-Native English Speakers. In *30th Workshop of the Psychology of Programming Interest Group - PPIG '19.*

[5] Brett A. Becker, Daniel Gallagher, Paul Denny, James Prather, Colleen Gostomski, Kelli Norris, and Garrett Powell. 2022. From the Horse's Mouth: The Words We Use to Teach Diverse Student Groups Across Three Continents. In *Proc. of the 53rd ACM Technical Symposium on Computer Science Education - Volume 1.* ACM, NY, NY, USA, 71–77.

[6] Seth Bernstein, Paul Denny, Juho Leinonen, Lauren Kan, Arto Hellas, Matt Littlefield, Sami Sarsa, and Stephen Macneil. 2024. "Like a Nesting Doll": Analyzing Recursion Analogies Generated by CS Students Using Large Language Models. In *Proc. of the 2024 on Innovation and Technology in Computer Science Education V. 1.* ACM, NY, NY, USA, 122–128.

[7] Virginia Braun and Victoria Clarke. 2019. Reflecting on Reflexive Thematic Analysis. *Qualitative Research in Sport, Exercise and Health* 11, 4 (2019), 589–597.

[8] Merijke Coenraad, Jen Palmer, Donna Eatinger, David Weintrop, and Diana Franklin. 2022. Using participatory design to integrate stakeholder voices in the creation of a culturally relevant computing curriculum. *Int. J. of Child-Computer Interaction* 31 (2022), 100353.

[9] Malcolm Corney, Sue Fitzgerald, Brian Hanks, Raymond Lister, Renee McCauley, and Laurie Murphy. 2014. 'Explain in Plain English' Questions Revisited: Data Structures Problems. In *Proc. of the 45th ACM Technical Symposium on Computer Science Education.* ACM, NY, NY, USA, 591–596.

[10] Raena Cota, Enrico Pontelli, Paige Prescott, Lauren Curry, Lisa Hufstedler, Francis Vigil, Yolanda Lozano, and David Rutledge. 2022. Culturally Responsive Pedagogy in Computer Science (CR in CS)- K-12 Teacher Professional Development- Needs and Challenges. In *Proc. of the 53rd ACM Technical Symposium on Computer Science Education V. 2.* ACM, NY, NY, USA, 1187.

[11] Andre Del Carpio Gutierrez, Paul Denny, and Andrew Luxton-Reilly. 2024. Evaluating Automatically Generated Contextualised Programming Exercises. In *Proc. of the 55th ACM Technical Symposium on Computer Science Education V. 1.* ACM, NY, NY, USA, 289–295.

[12] Paul Denny, Viraj Kumar, and Nasser Giacaman. 2023. Conversing with Copilot: Exploring Prompt Engineering for Solving CS1 Problems Using Natural Language. In *Proc. of the 54th ACM Technical Symposium on Computer Science Education V. 1.* ACM, NY, USA, 1136–1142.

[13] Paul Denny, Juho Leinonen, James Prather, Andrew Luxton-Reilly, Thezyrie Amarouche, Brett A. Becker, and Brent N. Reeves. 2024. Prompt Problems: A New Programming Exercise for the Generative AI Era. In *Proc. of the 55th ACM Technical Symposium on Computer Science Education V. 1.* ACM, NY, NY, USA, 296–302.

[14] Paul Denny, James Prather, Brett A. Becker, James Finnie-Ansley, Arto Hellas, Juho Leinonen, Andrew Luxton-Reilly, Brent N. Reeves, Eddie Antonio Santos, and Sami Sarsa. 2024. Computing Education in the Era of Generative AI. *Commun. ACM* 67, 2 (jan 2024), 56–67.

[15] Paul Denny, David H. Smith, Max Fowler, James Prather, Brett A. Becker, and Juho Leinonen. 2024. Explaining Code with a Purpose: An Integrated Approach for Developing Code Comprehension and Prompting Skills. In *Proc. of the 2024 on Innovation and Technology in Computer Science Education V. 1.* ACM, NY, NY, USA, 283–289.

[16] Krishno Dey, Prerona Tarannum, Md. Arid Hasan, Imran Razzak, and Usman Naseem. 2024. Better to Ask in English: Evaluation of Large Language Models on English, Low-resource and Cross-Lingual Settings. arXiv:2410.13153 [cs.CL]

[17] Jacob Doughty, Zipiao Wan, Anishka Bompelli, Jubahed Qayum, Taozhi Wang, Juran Zhang, Yujia Zheng, Aidan Doyle, Pragnya Sridhar, Arav Agarwal, et al. 2024. A comparative study of AI-generated (GPT-4) and human-crafted MCQs in programming education. In *Proc. of the 26th Australasian Computing Education Conf.* 114–123.

[18] Tony Haoran Feng, Paul Denny, Burkhard C Wünsche, Andrew Luxton-Reilly, and Jacqueline Whalley. 2024. An Eye for an AI: Evaluating GPT-4o's Visual Perception Skills and Geometric Reasoning Skills Using Computer Graphics Questions. In *SIGGRAPH Asia 2024 Educator's Forum* (Tokyo, Japan) *(SA Educator's Forum '24).* Association for Computing Machinery, New York, NY, USA. https://doi.org/10.1145/3680533.3697064

[19] James Finnie-Ansley, Paul Denny, Brett A. Becker, Andrew Luxton-Reilly, and James Prather. 2022. The Robots Are Coming: Exploring the Implications of OpenAI Codex on Introductory Programming. In *Australasian Computing Education Conf.* ACM, NY, NY, USA, 10–19.

[20] James Finnie-Ansley, Paul Denny, Andrew Luxton-Reilly, Eddie Antonio Santos, James Prather, and Brett A. Becker. 2023. My AI Wants to Know If This Will Be on the Exam: Testing OpenAI's Codex on CS2 Programming Exercises. In *Proc. of the 25th Australasian Computing Education Conf.* ACM, NY, NY, USA, 97–104.

[21] Patricia I. Fusch and Lawrence R. Ness. 2015. Are we there yet? Data saturation in qualitative research. (2015).

[22] Philip J. Guo. 2018. Non-Native English Speakers Learning Computer Programming: Barriers, Desires, and Design Opportunities. In *Proc. of the 2018 CHI Conf. on Human Factors in Computing Systems.* ACM, NY, NY, USA, 1–14.

[23] Carmen Nayeli Guzman, Anne Xu, and Adalbert Gerald Soosai Raj. 2021. Experiences of Non-Native English Speakers Learning Computer Science in a US University. In *Proc. of the 52nd ACM Technical Symposium on Computer Science Education.* ACM, NY, NY, USA, 633–639.

[24] Scott Hanselman. 2008. Do You Have To Know English To Be A Programmer? https://www.hanselman.com/blog/do-you-have-to-know-english-to-be-a-programmer Accessed: 2024-10-10.

[25] Arto Hellas, Juho Leinonen, Sami Sarsa, Charles Koutcheme, Lilja Kujanpää, and Juha Sorva. 2023. Exploring the Responses of Large Language Models to Beginner Programmers' Help Requests. In *Proc. of the 2023 ACM Conf. on Int. Computing Education Research - Volume 1.* ACM, NY, NY, USA, 93–105.

[26] Muntasir Hoq, Yang Shi, Juho Leinonen, Damilola Babalola, Collin Lynch, Thomas Price, and Bita Akram. 2024. Detecting ChatGPT-generated code submissions in

a CS1 course using machine learning models. In *Proc. of the 55th ACM Technical Symposium on Computer Science Education V. 1*. 526–532.

[27] Irene Hou, Owen Man, Sophia Mettille, Sebastian Gutierrez, Kenneth Angelikas, and Stephen MacNeil. 2024. More Robots are Coming: Large Multimodal Models (ChatGPT) can Solve Visually Diverse Images of Parsons Problems. In *Proc. of the 26th Australasian Computing Education Conf.* ACM, NY, NY, USA, 29–38.

[28] Irene Hou, Sophia Mettille, Owen Man, Zhuo Li, Cynthia Zastudil, and Stephen MacNeil. 2024. The Effects of Generative AI on Computing Students' Help-Seeking Preferences. In *Proc. of the 26th Australasian Computing Education Conf.* 39–48.

[29] David H. Smith IV, Viraj Kumar, and Paul Denny. 2024. Explain in Plain Language Questions with Indic Languages: Drawbacks, Affordances, and Opportunities. arXiv:2409.20297 [cs.CY]

[30] Sharin Jacob, Leiny Garcia, and Mark Warschauer. 2020. *Leveraging Multilingual Identities in Computer Science Education.* Springer Int. Publishing, 309–331.

[31] Mollie Jordan, Kevin Ly, and Adalbert Gerald Soosai Raj. 2024. Need a Programming Exercise Generated in Your Native Language? ChatGPT's Got Your Back: Automatic Generation of Non-English Programming Exercises Using OpenAI GPT-3.5. In *Proc. of the 55th ACM Technical Symposium on Computer Science Education V. 1*. ACM, NY, NY, USA, 618–624.

[32] Breanna Jury, Angela Lorusso, Juho Leinonen, Paul Denny, and Andrew Luxton-Reilly. 2024. Evaluating LLM-generated Worked Examples in an Introductory Programming Course. In *Proceedings of the 26th Australasian Computing Education Conference* (Sydney, NSW, Australia) *(ACE '24)*. Association for Computing Machinery, New York, NY, USA, 77–86. https://doi.org/10.1145/3636243.3636252

[33] Chris Kerslake, Paul Denny, David H. Smith IV, James Prather, Juho Leinonen, Andrew Luxton-Reilly, and Stephen MacNeil. 2024. Integrating Natural Language Prompting Tasks in Introductory Programming Courses. In *SIGCSE Virtual 2024.*

[34] Natalie Kiesler, Dominic Lohr, and Hieke Keuning. 2023. Exploring the Potential of Large Language Models to Generate Formative Programming Feedback. *arXiv preprint arXiv:2309.00029* (2023).

[35] Charles Koutcheme, Nicola Dainese, Sami Sarsa, Arto Hellas, Juho Leinonen, and Paul Denny. 2024. Open Source Language Models Can Provide Feedback: Evaluating LLMs' Ability to Help Students Using GPT-4-As-A-Judge. In *Proc. of the 2024 on Innovation and Technology in Computer Science Education V. 1*. 52–58.

[36] Viraj Kumar. 2021. Refute: An Alternative to 'Explain in Plain English' Questions. In *Proc. of the 17th ACM Conf. on Int. Computing Education Research*. ACM, NY, NY, USA, 438–440.

[37] Juho Leinonen, Paul Denny, Stephen MacNeil, Sami Sarsa, Seth Bernstein, Joanne Kim, Andrew Tran, and Arto Hellas. 2023. Comparing Code Explanations Created by Students and Large Language Models. In *Proc. of the 2023 Conf. on Innovation and Technology in Computer Science Education V. 1*. ACM, NY, NY, USA, 124–130.

[38] Juho Leinonen, Arto Hellas, Sami Sarsa, Brent Reeves, Paul Denny, James Prather, and Brett A. Becker. 2023. Using Large Language Models to Enhance Programming Error Messages. In *Proc. of the 54th ACM Technical Symposium on Computer Science Education V. 1*. ACM, NY, NY, USA, 563–569.

[39] Hayley C. Leonard and Sue Sentance. 2021. Culturally-relevant and responsive pedagogy in computing: A Quick Scoping Review. *Int. J. of Computer Science Education in Schools* 5, 2 (2021), 3–13.

[40] Raymond Lister, Colin Fidge, and Donna Teague. 2009. Further Evidence of a Relationship between Explaining, Tracing and Writing Skills in Introductory Programming. In *Proc. of the 14th Annual ACM SIGCSE Conf. on Innovation and Technology in Computer Science Education*. ACM, NY, NY, USA, 161–165.

[41] Chaoqun Liu, Wenxuan Zhang, Yiran Zhao, Anh Tuan Luu, and Lidong Bing. 2024. Is Translation All You Need? A Study on Solving Multilingual Tasks with Large Language Models. arXiv:2403.10258 [cs.CL]

[42] Evanfiya Logacheva, Arto Hellas, James Prather, Sami Sarsa, and Juho Leinonen. 2024. Evaluating Contextually Personalized Programming Exercises Created with Generative AI. In *Proc. of the 2024 ACM Conf. on Int. Computing Education Research - Volume 1*, Vol. 1. ACM, NY, NY, USA, 95–113.

[43] Mike Lopez, Jacqueline Whalley, Phil Robbins, and Raymond Lister. 2008. Relationships between Reading, Tracing and Writing Skills in Introductory Programming. In *Proc. of the Fourth Int. Workshop on Computing Education Research*. ACM, NY, NY, USA, 101–112.

[44] Stephen MacNeil, Andrew Tran, Arto Hellas, Joanne Kim, Sami Sarsa, Paul Denny, Seth Bernstein, and Juho Leinonen. 2023. Experiences from Using Code Explanations Generated by Large Language Models in a Web Software Development E-Book. In *Proc. of the 54th ACM Technical Symposium on Computer Science Education V. 1*. ACM, NY, NY, USA, 931–937.

[45] Alexandre Magueresse, Vincent Carles, and Evan Heetderks. 2020. Low-resource Languages: A Review of Past Work and Future Challenges. arXiv:2006.07264 [cs.CL]

[46] Gretchen McCulloch. 2019. Coding Is for Everyone–as Long as You Speak English. https://www.wired.com/story/coding-is-for-everyoneas-long-as-you-speak-english/ Accessed: 2024-10-10.

[47] Ismael Villegas Molina, Audria Montalvo, Benjamin Ochoa, Paul Denny, and Leo Porter. 2024. Leveraging LLM Tutoring Systems for Non-Native English Speakers in Introductory CS Courses. arXiv:2411.02725 [cs.HC] https://arxiv.org/abs/2411.02725

[48] Michael Sheinman Orenstrakh, Oscar Karnalim, Carlos Anibal Suarez, and Michael Liut. 2024. Detecting LLM-generated text in computing education: Comparative study for ChatGPT cases. In *2024 IEEE 48th Annual Computers, Software, and Applications Conf.* IEEE, 121–126.

[49] James Prather, Paul Denny, Juho Leinonen, Brett A. Becker, Ibrahim Albluwi, Michelle Craig, Hieke Keuning, Natalie Kiesler, Tobias Kohn, Andrew Luxton-Reilly, Stephen MacNeil, Andrew Petersen, Raymond Pettit, Brent N. Reeves, and Jaromir Savelka. 2023. The Robots Are Here: Navigating the Generative AI Revolution in Computing Education. In *Proc. of the 2023 Working Group Reports on Innovation and Technology in Computer Science Education*. ACM, NY, NY, USA, 108–159.

[50] James Prather, Raymond Pettit, Kayla McMurry, Alani Peters, John Homer, and Maxine Cohen. 2018. Metacognitive Difficulties Faced by Novice Programmers in Automated Assessment Tools. In *Proc. of the 2018 ACM Conf. on Int. Computing Education Research*. ACM, NY, NY, USA, 41–50.

[51] James Prather, Brent N Reeves, Juho Leinonen, Stephen MacNeil, Arisoa S Randrianasolo, Brett A. Becker, Bailey Kimmel, Jared Wright, and Ben Briggs. 2024. The Widening Gap: The Benefits and Harms of Generative AI for Novice Programmers. In *Proc. of the 2024 ACM Conf. on Int. Computing Education Research - Volume 1*. ACM, NY, NY, USA, 469–486.

[52] Adalbert Gerald Soosai Raj, Kasama Ketsuriyonk, Jignesh M. Patel, and Richard Halverson. 2017. What Do Students Feel about Learning Programming Using Both English and Their Native Language?. In *Int. Conf. on Learning and Teaching in Computing and Engineering*. 1–8.

[53] Brent Reeves, Sami Sarsa, James Prather, Paul Denny, Brett A. Becker, Arto Hellas, Bailey Kimmel, Garrett Powell, and Juho Leinonen. 2023. Evaluating the Performance of Code Generation Models for Solving Parsons Problems With Small Prompt Variations. In *Proc. of the 2023 Conf. on Innovation and Technology in Computer Science Education V. 1*. ACM, NY, NY, USA, 299–305.

[54] Eddie Antonio Santos and Brett A Becker. 2024. Not the Silver Bullet: LLM-enhanced Programming Error Messages are Ineffective in Practice. *arXiv preprint arXiv:2409.18661* (2024).

[55] Sami Sarsa, Paul Denny, Arto Hellas, and Juho Leinonen. 2022. Automatic Generation of Programming Exercises and Code Explanations Using Large Language Models. In *Proc. of the 2022 ACM Conf. on Int. Computing Education Research - Volume 1*. ACM, NY, NY, USA, 27–43.

[56] Jaromir Savelka, Arav Agarwal, Marshall An, Chris Bogart, and Majd Sakr. 2023. Thrilled by Your Progress! Large Language Models (GPT-4) No Longer Struggle to Pass Assessments in Higher Education Programming Courses. *The 19th ACM Conf. on Int. Computing Education Research* (2023).

[57] David H. Smith, Paul Denny, and Max Fowler. 2024. Prompting for Comprehension: Exploring the Intersection of Explain in Plain English Questions and Prompt Writing. In *Proc. of the Eleventh ACM Conf. on Learning @ Scale*. ACM, NY, NY, USA, 39–50.

[58] David H. Smith and Craig Zilles. 2024. Code Generation Based Grading: Evaluating an Auto-grading Mechanism for "Explain-in-Plain-English" Questions. In *Proc. of the 2024 on Innovation and Technology in Computer Science Education V. 1*. ACM, NY, NY, USA, 171–177.

[59] Adalbert Gerald Soosai Raj, Kasama Ketsuriyonk, Jignesh M. Patel, and Richard Halverson. 2018. Does Native Language Play a Role in Learning a Programming Language?. In *Proc. of the 49th ACM Technical Symposium on Computer Science Education*. ACM, NY, NY, USA, 417–422.

[60] Chris Stokel-Walker. 2024. *AI chatbot models 'think' in English even when using other languages.* https://www.newscientist.com/article/2420973-ai-chatbot-models-think-in-english-even-when-using-other-languages/

[61] Andrew Taylor, Alexandra Vassar, Jake Renzella, and Hammond Pearce. 2024. dcc --help: Transforming the Role of the Compiler by Generating Context-Aware Error Explanations with Large Language Models. In *Proc. of the 55th ACM Technical Symposium on Computer Science Education V. 1*. 1314–1320.

[62] Andrew Tran, Kenneth Angelikas, Egi Rama, Chiku Okechukwu, David H. Smith, and Stephen MacNeil. 2023. Generating multiple choice questions for computing courses using large language models. In *2023 IEEE Frontiers in Education Conf.* IEEE, 1–8.

[63] Marcella Veldthuis and Felienne Hermans. 2024. A Word about Programming: Applying a Natural Language Vocabulary Acquisition Model to Programming Education. In *Proc. of the 2024 ACM SIGPLAN Int. Symposium on SPLASH-E*. ACM, NY, NY, USA, 56–65.

[64] Anne Venables, Grace Tan, and Raymond Lister. 2009. A Closer Look at Tracing, Explaining and Code Writing Skills in the Novice Programmer. In *Proc. of the Fifth Int. Workshop on Computing Education Research Workshop*. ACM, NY, NY, USA, 117–128.

[65] Sierra Wang, John Mitchell, and Chris Piech. 2024. A large scale RCT on effective error messages in CS1. In *Proc. of the 55th ACM Technical Symposium on Computer Science Education V. 1*. 1395–1401.

[66] Jacqueline L. Whalley, Raymond Lister, Errol Thompson, Tony Clear, Phil Robbins, P. K. Ajith Kumar, and Christine Prasad. 2006. An Australasian study of reading and comprehension skills in novice programmers, using the bloom and SOLO

taxonomies. In *Proc. of the 8th Australasian Conf. on Computing Education - Volume 52*. Australian Computer Society, Inc., AUS, 243–252.

[67] Jun Zhao, Zhihao Zhang, Luhui Gao, Qi Zhang, Tao Gui, and Xuanjing Huang. 2024. LLaMA Beyond English: An Empirical Study on Language Capability Transfer. arXiv:2401.01055 [cs.CL]