THE UNIVERSITY OF CHICAGO

UNPACKING HOW HUMANS COMMUNICATE PROGRAMMING TASKS IN NATURAL LANGUAGE TO SUPPORT END-USER PROGRAMMING WITH LLMS

A DISSERTATION SUBMITTED TO
THE FACULTY OF THE DIVISION OF THE PHYSICAL SCIENCES DIVISION
IN CANDIDACY FOR THE DEGREE OF
MASTER OF SCIENCE

DEPARTMENT OF COMPUTER SCIENCE

BY
MADISON PICKERING

CHICAGO, ILLINOIS
MAY 2024

This is dedicated to my friends, family, and cat who respectively supported me, checked on me, and screamed at me for food during the course of this project.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGMENTS

I would like to acknowledge my collaborators: Alison Gan, Helena Williams, Weijia He, Michael Littman, and my advisor Blase Ur, for all of their work and guidance. Similarly, many other members of UChicago's SUPERgroup lab provided valuable feedback on various portions of this project (in reverse alphabetical order): Emma Peterson, Elleen Pan, Kevin Bryson, and Arthur Borem. My primary contributions to this thesis are in drafting large portions of the paper; implementation of the experimental infrastructure back-end and LLM querying, testing; codebook development; and performing the bulk of the analysis.

ABSTRACT

Large language models (LLMs) like GPT-4 and Code Llama can convert natural-language descriptions into computer code, inspiring many people to see them as a promising interface for end-user programming. We undertake a systematic analysis of how people without programming experience describe information-processing tasks (IPTs) in natural language, focusing on the characteristics that make descriptions successful. In an online study, we paired 242 crowdworkers and asked one member of each pair (the sender) to communicate IPTs to the other (the receiver) in natural language. Participants answered test cases (producing outputs for given inputs) based on these descriptions. We also fed these same descriptions to six LLMs to see how they would answer the test cases. We found that participants both with and without prior programming experience successfully communicated IPTs, though descriptions written by senders with programming experience enabled both human receivers and LLMs to answer more test cases correctly. Descriptions containing example test cases led human receivers to answer 16.8% more test cases correctly, with similar gains by LLMs. Receivers who asked certain clarifying questions (e.g., about edge cases) answered slightly more test cases correctly. The best LLMs we tested narrowly outperformed humans in answering test cases.

CHAPTER 1

INTRODUCTION

Large Language Models (LLMs) are demonstrating remarkable gains in domain-specific task performance, resulting in their rapid adoption. One such domain is writing computer code. These models can generate entire functions when prompted. That is, LLMs enable humans to write *code* using only *natural language* prompts. For example, a skilled engineer may prompt a model as follows: "*Compare the number of A's and B's before the letter C and report which has more. When tied or no votes, report A.*" The model would then output lines of code that attempt to accomplish that task.

However, there is no guarantee that a human, especially one without any programming experience, will communicate a task in a way that leads to successful code. Consider a non-programmer attempting to communicate the same task: "*Count the occurrence of letters "A" and "B". It is not necessary to count the C's. If A = B or A > B, then you report "A". Otherwise, report "B". If a equals b or a is greater than b then report a.*" While both versions are technically correct, the programming expert's version is more clear and precise.

We imagine a future in which LLMs enable end-user programming (i.e., by users without any prior programming experience) in natural language. To this end, we investigated how humans communicate computational tasks using only natural language as an interface. Through a between-subjects online study, we compare how participants with and without programming experience differ when communicating such tasks in natural language, in addition to examining facets and factors that improve this communication.

To be precise, this paper focuses on **Information Processing Tasks (IPTs)**, which are tasks that necessitate doing a calculation that involves a self-contained formal transformation of input to output. The A/B task above is an IPT. Another example is, "You will be given a list containing numbers. Report the sum of the positive elements." In our study, the communication of an IPT involves two parties: a **sender**, who substantially rephrases four different IPTs we provide, and a **receiver**, who interprets the sender's description. To measure understanding, we have both the sender and receiver answer **test cases** (specifying the output for given inputs).

As part of our study, we randomly assign two elements that we hypothesized would improve IPT communication. One is the inclusion of **examples**, which means the participant includes appropriate input/output pairs for a given task when describing an IPT in natural language. We also randomly assigned some sender-receiver pairs the ability to **interact** via a chat window to clarify the IPT. Because we focused on the ability of end users to program via natural-language communication, we further distinguish between **end users (non-programmers)** and **programmers**. We consider a programmer to be any participant who identifies as a programmer and can answer simple programming questions (Danilova et al. [2021]). We also qualitatively explored the facets of successful IPT communication.

To this end, we investigated a number of research questions. **RQ 1:** What role does *programming experience* play in successful IPT communication? Further, **RQ 1a:** Can non-programmers successfully communicate an IPT? We found that individuals with programming experience tended to be more successful at communicating IPTs. However, the difference in overall performance on test cases was relatively small (8.1%). We believe this suggests that while programming experience does help with IPT communication, the difference is not so stark as to suggest that non-programmers cannot successfully communicate IPTs. We then delved into why programmers do better than non-programmers at communicating IPTs. **RQ 1b** asks what programmers do that end users do not do during IPT communication. We find that programmers are more likely to include examples when prompted than non-programmers. Additionally, programmers were less likely to interact or request clarification than non-programmers.

We also investigated the effect of examples. **RQ 2:** What role do *examples* play in successful IPT communication? We found that examples play a large, beneficial role in successful IPT communication. Receivers whose corresponding sender included examples for at least half of their assigned IPTs performed 18% better in answering test cases than those who did not. However, we found that participants sometimes gave incorrect examples, impairing communication. We consequently investigated **RQ 2a:** What types of examples benefit communication? We found that examples that were entirely original (i.e., not reused from what we provided) and entirely correct correlated the strongest with successful IPT communication. We also asked **RQ 2b:** Are there differences in the examples that programmers versus end-users include? We found that programmers were more likely to come up with their own examples than non-programmers, and less likely to include explanations along with their examples or add additional formatting.

Given that interaction (traditionally with a compiler) is an integral part of effective programming, we investigated **RQ 3:** What types of *interactions* are beneficial when communicating a task in English prose? We ultimately found that the vast majority of receivers' requests for clarification did not correlate with positive outcomes. However, requests to clarify certain variables in the IPT, or the IPT's expected input and output, were more likely to result in positive outcomes, such as additional relevant information.

Finally, we investigated **RQ 4**: To what degree do the successful characteristics of a natural language IPT description for humans also apply to LLMs? As such, we fed our human senders' natural language IPT descriptions to versions of popular LLMs: GPT-4, Chat Llama, Code Llama, and Gemini. Like human receivers, LLMs performed best when the descriptions included examples and, to a less extent, when they were written by a sender with programming experience. If a human receiver did not fully understand the sender's description, there was a 73.3% chance that the LLM would also be incorrect. Notably, GPT-4 performed comparably to a human receiver, and all but one of the LLMs we tested were more successful in directly answering test cases than producing code to do so.

We conclude by noting that because the relative difference in successful IPT communication is small between programmers and non-programmers, certain interventions may be able to bridge this gap. Namely, designing interfaces that promote the inclusion of examples and encourage specific types of clarification may help non-programmers communicate IPTs as successfully as individuals with programming experience.

CHAPTER 2

RELATED WORK

While there is little prior work on end-user programming using LLMs as a natural language interface, we discuss related work on LLM-based tools for experienced programmers, other interfaces for end-user programming, and the use of LLMs as educational tools.

2.1 End-User Programming

Over the past few decades, there has been substantial interest in designing systems and interfaces that empower end users without training or experience in computer programming to express computational tasks (Nardi [1993], Lieberman et al. [2006]). Many systems for end-user programming rely on graphical user interfaces and visual programming (Myers et al. [2006]) rather than forcing users to memorize complex syntax. For instance, the Blockly (Fraser [2015]) and Scratch (Resnick et al. [2009]) languages let users express programs by connecting blocks and include common programming concepts like loops and conditionals, but presented graphically (Resnick et al. [2009]). Similarly, trigger-action programming gives users a graphical user interface for defining event-driven rules (Ur et al. [2014], Huang and Cakmak [2015]) and has been popularized in recent years via commercial services like Zapier (Rahmati et al. [2017]) and IFTTT (Mi et al. [2017], Ur et al. [2016]). Current instantiations of trigger-action programming enable end users to define automations in domains ranging from smart homes (Ur et al. [2014]) to robotics (Leonardi et al. [2019]) to social media (Ur et al. [2016]). Other systems, such as prototypes for creating web mashups (Wong and Hong [2007]), ask users to specify workflows graphically. More broadly, a recent literature survey (Barricelli et al. [2019]) noted over a dozen types of end-user programming interfaces discussed in the literature, including gesture-based, template-based, and spreadsheet-based interfaces. Another recent literature survey (Kuhail et al. [2021]) characterized these interfaces more abstractly as block-based, icon-based, form-based, or diagram-based.

While natural language has long been discussed as an interface for end-user programming (Barricelli et al. [2019]), we expect that the recent advances in LLMs may reflect an inflection point in enabling natural language to be practical in general domains for end-user programming. Thus, we focus on how non-technical end users and trained programmers express IPTs in natural language, evaluating what characteristics of the sender, the description they write, and the instructions they were given impact the ability of humans and LLMs to interpret these descriptions. To our knowledge, prior work has not focused on LLM-based natural language interfaces for end-user programming, though concurrent work prototyped a system for doing so in the narrower context of robot programming (Karli et al. [2024]).

Programming by demonstration (PbD) or by example has also long been studied as a type of end-user programming (Myers et al. [2006]). While not our main focus, we test a condition in which senders are encouraged to augment their natural language description with example test cases, following the PbD paradigm.

2.2 Code Generators

In this section, we discuss LLMs used for code generation, then cover relevant related work for code-generating LLMs.

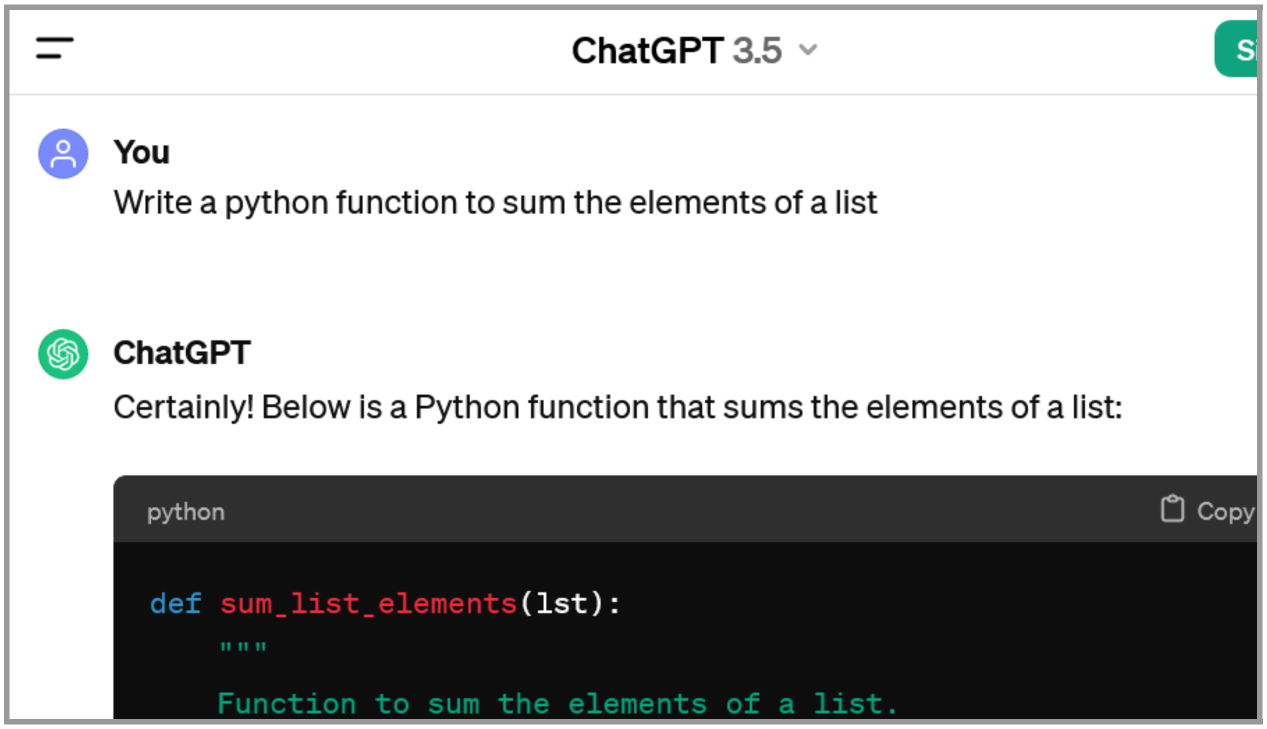
2.2.1 Background

The relationship between code and general natural language was brought to the forefront of research attention during the development of GPT-3 (Brown et al. [2020]). Researchers found that after being exposed to code during training, the model could produce *new* code with startling clarity. This exciting result motivated researchers at OpenAI to further train, or fine-tune, GPT-3 on public GitHub repositories. This resulted in *Codex*, an LLM specifically designed to generate code (Chen et al. [2021]). Codex has since been incorporated as the backend of Github Copilot, and iterated upon considerably. Other companies like Google and Meta have similarly produced their own LLMs capable of high performance on coding tasks: Gemini and Code Llama, respectively (Google [2023], Rozière et al. [2024]).

Modern LLMs work by producing the most likely next bit of text, given an input string (Radford et al. [2019], Brown et al. [2020], OpenAI [2022], Touvron et al. [2023], Google [2023]). This enables humans to interact with, or **prompt**, LLMs by supplying them with input strings. Because code-generating LLMs are developed by taking a model trained on both natural language and code, the models can be prompted with *both* natural language and/or code. For example, a user may prompt the model with "Write a line of python that checks if a number is even or odd", *or* "`is_even = `" to which the model would produce the appropriate completion. Prompting has led to LLMs being coupled with a UI and adopted as interactive tools. These tools may be fully integrated into an IDE, such as Github Copilot, or stand alone as web apps in the case of ChatGPT (Figure 2.1) and Code Llama (Figure 2.2).

2.2.2 LLMs As Programming Support Tools

Understanding LLMs as programming support tools has primarily been done through two lenses: characterizing the human-LLM interactions themselves (to find more useful interfaces), or characterizing the correctness of produced code. Taking the former route, Lee et al. develop an open-ended framework to understand human-LLM interactions across various tasks (Lee et al. [2024]). However, many others adopt a more grounded approach towards understanding human-LLM interactions. Liu et al., for example, develop a method of improving natural language input to an LLM, called grounded abstraction matching, where a natural language description is mapped to specific system actions and then returned to the user for review (Liu et al. [2023]). McNutt et al. similarly map open-ended NLP to specific outcomes by characterizing the potential design space for notebook-based code as-



You

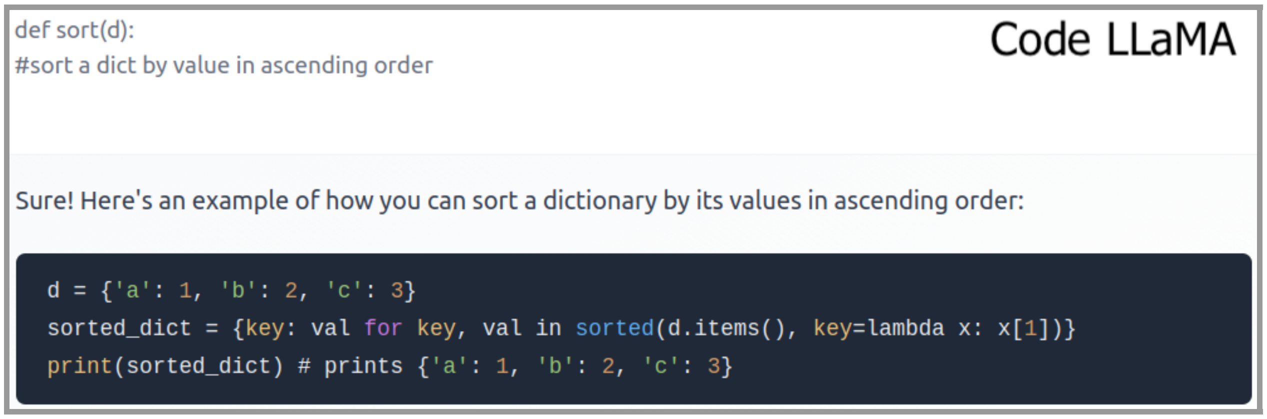
Write a python function to sum the elements of a list

ChatGPT

Certainly! Below is a Python function that sums the elements of a list:

```python
def sum_list_elements(lst):
    """
    Function to sum the elements of a list.
```

Figure 2.1: ChatGPT allows for a chatbot-based interface.



```
def sort(d):
#sort a dict by value in ascending order
```

Sure! Here's an example of how you can sort a dictionary by its values in ascending order:

```python
d = {'a': 1, 'b': 2, 'c': 3}
sorted_dict = {key: val for key, val in sorted(d.items(), key=lambda x: x[1])}
print(sorted_dict) # prints {'a': 1, 'b': 2, 'c': 3}
```

Figure 2.2: Meta Code Llama's interface is a widget that delineates between the prompt (top) and response (bottom).

sistants (McNutt et al. [2023]). The authors note that the most useful interfaces differ based on the task at hand. Ross et al. primarily investigate user satisfaction under different interfaces (Ross et al. [2023]). Echoing the idea of self perception as a productivity metric, (Vaithilingam et al. [2022], Kalliamvakou [2022], Meyer et al. [2021]) respectively investigate perceptions of productivity when developers use LLMs as programming support tools, and/or perceptions of productivity despite the fact that LLMs may *not* in fact improve the task completion time or success rate.

Others have taken the approach of investigating LLM correctness during programming. The de-facto method of determining code-generator correctness is *functional correctness*, which essentially states that code is correct if it passes associated unit tests (Chen et al. [2021], Google [2023], Rozière et al. [2024]). Model developer have primarily been investigating code-correcteness during model development, however, others have been focusing on the quality of produced code post model development (Tambon et al. [2024], Dakhel et al. [2023], Döderlein et al. [2023], Nguyen and Nadi [2022]). Others investigate the tendency for LLM code-generators to produce insecure code (Sandoval et al. [2023], Pearce et al. [2021], Perry et al. [2022]), paying special attention to GitHub Copilot. The authors generally find that the LLM does produce insecure code, yet the model does not introduce security vulnerabilities at a rate greater than humans (Sandoval et al. [2023]).

2.2.3 Programming Education

Prior studies have also explored the application of code generators to programming education, especially amid concerns that students may use LLMs to complete their assignments (Roose [2024]). Approaches have stemmed from evaluating code-generating LLMs on introductory programming tasks to determine the extent to which they may be used for either cheating or educational support tools (Finnie-Ansley et al. [2022, 2023], Sarsa et al. [2022]). Researchers generally found that a vast majority of auto-generated programming exercises and code explanations were sensible and ready to use except for minor mistakes that could easily be corrected by an instructor. These results imply great potential for code generators in creating introductory programming course material, whose abilities will likely improve over time. Jayagopal et al. similarly take an optimistic approach towards code-generating LLMs (Jayagopal et al. [2022b]). The authors investigate how LLMs may be more easily learnable as tools by novice programmers, for the purpose of facilitating programming education.

CHAPTER 3
METHODS

To better understand IPT communication in prose, we conducted an online user study. We recruited sets of participants who either did or did not have prior programming experience and had them complete a pre-survey. We then directed them to our custom-built web application, where they were paired with one another in real time. We assigned them to a condition indicating who would describe IPTs to the other, whether they were asked to add examples to their description, and whether they were permitted to interact with each other. Each pair communicated four IPTs, and each member of the pair answered four test cases for each. Finally, participants completed a post-survey. This section details each of these aspects.

3.1 Recruitment and Demographic Survey

We recruited 242 participants from the Prolific crowdworking service (Prolific [2023]). We required participants to be (1) 18+ years old, (2) live in the United States, (3) speak English fluently, and (4) have completed 20+ Prolific studies with a 95%+ approval rating. To gauge the effect of prior computer programming experience on IPT communication, we reserved half the slots on Prolific to users who indicated that they had programming experience when registering for Prolific, and half who indicated that they did not. We compensated participants \$8.00 USD if they completed the study, waited 20 minutes without being successfully paired (see Section 3.2), or if their assigned partner became unresponsive (as detected by our web app's interface, they did not move their mouse or type for ten minutes, or they closed the study window and did not rejoin within five minutes). Our study was approved by our university's IRB.

We then directed participants to an initial survey with standard demographic questions, which are reproduced in Appendix A.2.1.

3.2 Pairing and Conditions

At the end of the initial survey, we redirected participants to our web application with a unique login link. They stayed on a waiting screen until another participant joined, at which point they were paired. The first of the pair to land on the waiting screen was assigned the role of **sender**, who would describe IPTs in prose. The second was assigned the role of **receiver**, who would interpret the sender's description. If no partner was found within 20 minutes, the session was terminated and the participant was paid as if they had completed the study.

Our study had a between-subjects design. To answer RQ 2, we assigned each pair of participants round-robin to either the **interactive** condition, in which the receiver was permitted to ask clarifying questions of the sender about their IPT description in a chat window we provided, or the **non-interactive** condition, in which they were not. To answer RQ 3, we also randomly assigned each pair to either the **examples** condition, in which we both gave senders example test cases when presenting them each IPT and asked them to include different examples in their own description, or the **no-examples** condition, in which we did neither.

3.3 Primary Study Task and Interface

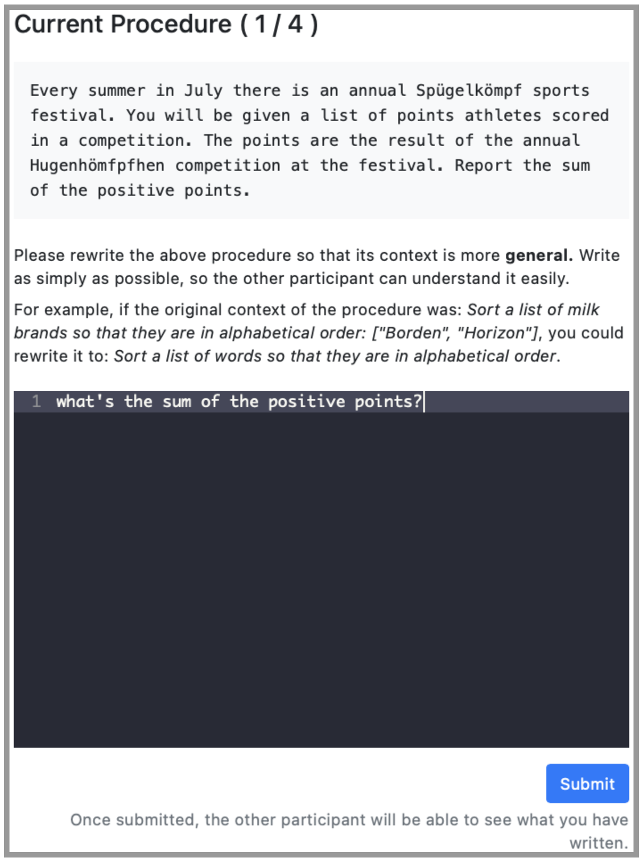
The main study task was for the sender to describe an IPT in prose to the receiver. A key challenge is that we needed to communicate a specific IPT to the sender in a way that would minimally influence the way they described that IPT in prose. While presenting an IPT in code and asking the sender to describe it in prose might be feasible if all senders were computer programmers, we specifically wanted to compare programmers and non-programmers.

Another option we considered was showing example inputs and their corresponding outputs, but we worried that it would be infeasible to communicate even moderately complex IPTs to senders by doing so.

Instead, we decided to give senders a verbose and context-specific prose description of each IPT in an interface element that rendered it as an image to prevent copy-pasting. We term this the **verbose description**. For instance, an IPT that was ultimately about returning the maximum value in a list had a verbose description about finding the largest number of days late someone could pay their rent when comparing apartments during a move to San Francisco. Our instructions to senders asked them to rewrite the task more simply and more generally (i.e., not specific to a context), which we termed the **sender's description**. We gave participants the sample of rewriting "sort a list of milk brands" as "sort a list of words." Section 3.4 describes our 12 IPTs and how we selected them.

After the sender and receiver were paired, our web application showed them a view customized to their role and condition. The sender's view (Figure 3.1) contained the verbose description of the IPT (gray box), the task instructions (below the gray box), a text editor in which the sender would rephrase the IPT (black), and a submit button. For pairs assigned to the examples condition, the verbose description also included example test cases (input/output pairs), and the instructions specifically asked the sender to include their own examples in their rephrasing.

The receiver's view (Figure 3.2) contained instructions, a (non-editable) text box where the sender's rephrased description appeared, and a button to accept the procedure. In the *interactive* condition, a second button for rejecting the description was also present. Rejecting the description opened a text box for the receiver to explain what they found unclear about the description. A chat window (Figure 3.3) then opened for both the sender and receiver to discuss this request for clarification interactively. The sender could then edit their description, pressing a button to resubmit it to the receiver. The receiver could


Current Procedure (1 / 4)

```
Every summer in July there is an annual Spügelkömpf sports
festival. You will be given a list of points athletes scored
in a competition. The points are the result of the annual
Hugenhömfpfhen competition at the festival. Report the sum
of the positive points.
```

Please rewrite the above procedure so that its context is more **general.** Write as simply as possible, so the other participant can understand it easily.

For example, if the original context of the procedure was: *Sort a list of milk brands so that they are in alphabetical order: ["Borden", "Horizon"]*, you could rewrite it to: *Sort a list of words so that they are in alphabetical order*.

```
what's the sum of the positive points?
```

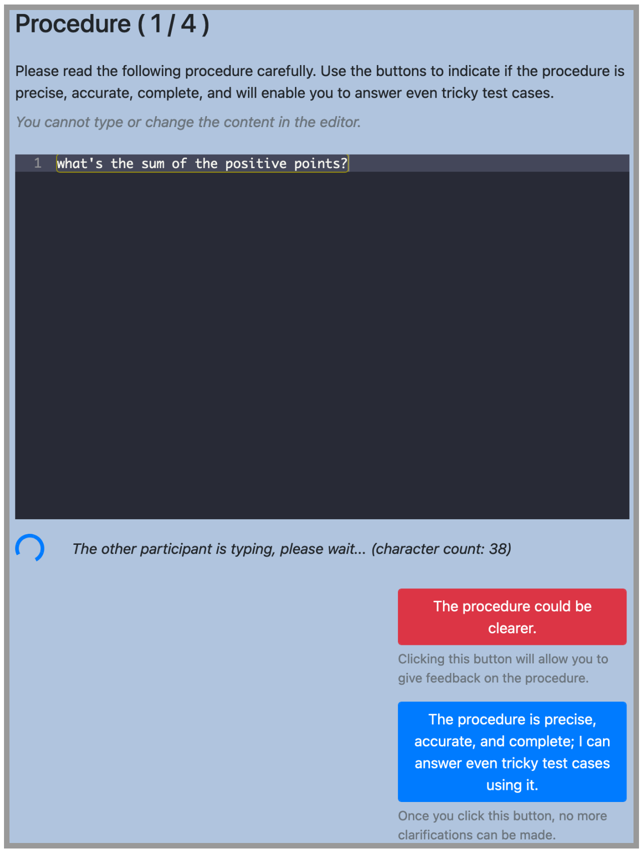
Once submitted, the other participant will be able to see what you have written.

Figure 3.1: The sender's view in the non-interactive condition.

again accept or reject the modified description, repeating as many times as the receiver felt necessary.

Once the receiver accepted the description, both the sender and receiver separately answered four **test cases**, specifying the expected output for four different inputs we provided. While the sender used the verbose description to answer these test cases, which were the same for both participants (and fixed for each IPT), the receiver used the sender's description. We termed two of the test cases **regular cases** because they tested common, standard behaviors (e.g., being asked to find the maximum given a well-formed list with a single largest number). We termed the other two test cases **edge cases** because they tested less standard behaviors (e.g., being asked to find the maximum when given an empty list or a list with multiple largest numbers). We chose to test both kinds of test cases to gauge how comprehensive and precise the sender's description was. The order of test cases was randomized.

After answering test cases, participants responded to two statements on 7-point Likert scales: *"I am confident my answers to the test cases are correct"* and *"I am confident I fully understand the procedure above."* We repeated this process for a total of four IPTs per pair with the roles (sender and receiver) and condition held constant.



Procedure (1 / 4)

Please read the following procedure carefully. Use the buttons to indicate if the procedure is precise, accurate, complete, and will enable you to answer even tricky test cases.

You cannot type or change the content in the editor.

```
what's the sum of the positive points?
```

The other participant is typing, please wait... (character count: 38)

The procedure could be clearer.
Clicking this button will allow you to give feedback on the procedure.
The procedure is precise, accurate, and complete; I can answer even tricky test cases using it.
Once you click this button, no more clarifications can be made.

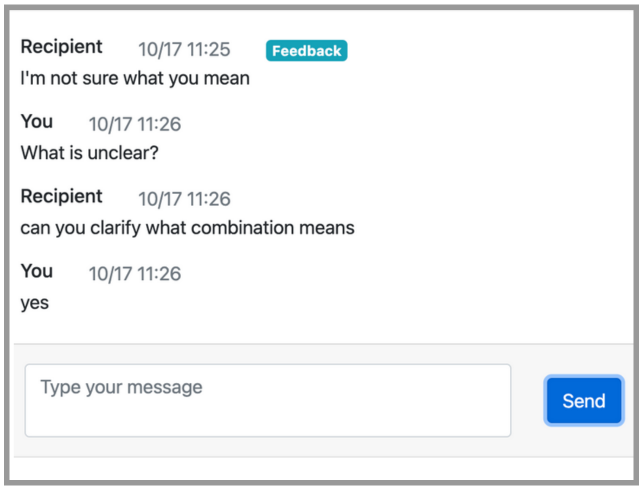
Figure 3.2: The receiver's view in the interactive condition.

3.4 Selection of IPTs

To inform problem selection, we examined IPTs from various sources, including four popular LLM benchmarks (HumanEval, APPS, MBPP, and BigBench datasets (Chen et al. [2021], Hendrycks et al. [2021], Austin et al. [2021], Srivastava et al. [2022]), a popular interview prep tool (LeetCode (LeetCode [2023])), and an introductory programming course (MIT

Table 3.1: Succinct descriptions of the 12 IPTs we tested. Appendix A.3 contains the (verbose, domain-specific) text given to senders.

Name	**Short Summary of the IPT**
`max`	Given a list of numbers, report the max.
`num_even`	Given a list of numbers, report how many even numbers are in the list.
`sum_pos`	Given a list of numbers, report the sum of the positive numbers in the list.
`reignfall`	Given a string containing As, Bs, and Cs, count the number of As and Bs, reading from left to right. Stop counting once a C is reached. If there are more Bs than As once the first C is reached, report B. Otherwise, report A.
`palindrome`	Report T if a string is a palindrome. Otherwise, report F.
`find_sum`	Given a list of numbers and a target number, report what pair of numbers from the list have a sum equal to the target. Report the larger number of the pair first. If there is no pair with sum equal to target, report [].
`rmv_dup`	Given a list, report the list after all instances of duplicates are removed, preserving the relative ordering.
`str_diff`	Given two strings, report which letter has been added to the first string to arrive at the second.
`is_subseq`	Given two strings, report T if the first string is a subsequence of the other. Otherwise, report F.
`exact_chg`	Given three numbers (Small, Big, Goal), report T if Small times 1 plus Big times 5 equals Goal. Otherwise, report F.
`fizzbuzz`	Given some number n, count and report the number of times the digit 7 appears in whole numbers less than n that are divisible by 11 or 13 (or both).
`xyz`	Given three numbers (x, y, z), if y > z, report the pair of numbers: [x + z, 0]. Otherwise, report the pair of numbers: [x + y, z - y].


Recipient 10/17 11:25
I'm not sure what you mean

You 10/17 11:26
What is unclear?

Recipient 10/17 11:26
can you clarify what combination means

You 10/17 11:26
yes

Figure 3.3: The chatbox shown in the interactive condition if a receiver rejects the sender's description.

(Canelake [2011])). We looked at these sources to understand how LLMs', experienced programmers', and beginner programmers' codes are judged. While we were inspired by the tasks from all corpora, we chose to ultimately only pull tasks directly from HumanEval, which was used to benchmark Codex (Chen et al. [2021]). We chose problems specifically from HumanEval as their tasks were generally self-contained and often did not require formal knowledge of data structures. Specifically, six of our tasks have been pulled from HumanEval, while the other six are of our own making.

IPTs were selected such that they varied in complexity, ranging from simply finding the maximum element in a list to describing a custom math function. Each IPT was then associated with four test cases, two of which were edge cases. We phrased each of the IPTs so as to eliminate computer science jargon. We further added text to contextualize each IPT, such as framing the problem "find the max of a list" as "Given a list of days you can pay your rent late, report the maximum number of days you can pay your rent late." We additionally added superfluous information to the IPT and asked participants to make it concise. This forces participants to meaningfully read and engage with the IPT during rephrasing. We finally note that while we did not consciously pull any problems from datasets other than

HumanEval, some of our problems exist in other datasets due to the generic nature of the tasks; an example is our eighth task, which asks if a string is a substring of another string (LeetCode [2023], Chen et al. [2021]). We believe this heavy overlap with problems in other testing suites is a good indicator that our selected IPTs are common and general, and may consequently reflect tasks that end-user programmers might do. All programming tasks and their associated test cases are shown in Appendix A.3, but a shorthand name and descriptor of those tasks are shown in Figure A.1.

3.5 Post-survey

Our post-survey primarily consists of questions relating to the participant's experience in the previous sections of the study, such as if they perceived the task to be difficult or not. We include all questions in the post-survey in Appendix A.2.2. However, we also ask in our post-survey questions to determine if a participant is a *programmer* or a *non-programmer* at this point. Prior literature notes that there exists a considerable discrepancy in the degree to which participants *self-report* programming knowledge and to what degree the participants actually *demonstrate* programming knowledge (Balebako et al. [2014]). We also observed this pattern, noting on many occasions that participants self-report that they have programming experience on Prolific, but indicate that they do *not* have programming experience when taking our post-survey.

We consequently used four questions to determine if a participant legitimately has programming knowledge. We used three of the screening questions developed by Danilova et al. [2021], and our fourth simply asks what programming languages the participant is proficient in. We consider the fourth question to be correct unless the participant has typed "none" there. Only participants who answer at least three questions correctly are counted as programmers.

3.6 Substituting LLMs For Human Receivers

To investigate how programmers and non-programmers communicate IPTs to LLMs, we performed an additional sub-study using the communications collected in the main experiment. Specifically, we took the final IPT wording as it was communicated from sender to receiver and prompted an LLM with it. To be clear, if participants had communicated and clarified the IPT in any way, we took the final phrasing of the IPT *post-communication* and fed it to an LLM. Because LLMs can either directly answer test cases based on the final phrasing or produce code, which can then be *run* on test cases, we prompt each LLM twice: once to directly answer test cases, once asking the LLM to produce a Python function that implements the IPT. Our prompt to directly answer test cases is identical to the prompt that we use for human receivers, while the prompt to produce code has an additional sentence: "Write a single Python function which is an implementation of the procedure above."

We specifically query GPT-4, Chat Llama, Code Llama, and Gemini due to their state-of-the-art performance (OpenAI [2024], et al. [2023], Google [2023], Rozière et al. [2024]). GPT-4 and Gemini purport to have strong ability in both NLP as well as code generation, but Llama 2 is split into two separate models: Chat Llama for natural language, and Code Llama for producing code (Touvron et al. [2023]). We use default system messages and querying parameters for all models, as provided in their documentation at the time of querying. When models are directly answering test cases, we ensure that the test cases follow the same order as the human receiver.

When the model produces code, it will sometimes produce a function that takes a different number of arguments than the input. In these cases, we first check if the function is particularly vulnerable to the order of arguments, such as IPT `xyz` (see Table 3.1). If so, we simply count the produced function as incorrect. Similarly, code that takes excessively long to execute ($>$10 seconds) or relies on input from the user (i.e., as through standard input) instead of its function parameters is considered to be incorrect. If the function is more or less agnostic to the order of its arguments, such as IPT `fizzbuzz` or `palindrome`, we combinatorially try all possibilities of formatting the test case such that the number of arguments in the test cases is consistent with the number of arguments in the produced function header. For example, if the test case took two inputs `x, y` and the function header only took one input, we attempt to format the input as `[x, y]`. We then run the produced code with all the possible input formats, and as long as one of the inputs produces the expected output, we count the test case as correct.

Finally, at least one researcher manually verifies the results from code execution to ensure that the produced code legitimately attempts to solve the test cases instead of simply guessing. Similar manual verification is performed to ensure that the correct output is being graded. This step is crucial when the LLM output includes function calls as usage examples, or has internal print statements in its function definition.

3.7 Data Analysis

The bulk of our quantitative analysis centered on linear regression models we built. The dependent variable (**DV**) was the number of test cases answered correctly by the sender, the receiver, or an LLM in place of the human receiver. These regression models' independent variables (**IVs**) included the assigned condition, the demographics and experiences of the sender and/or receiver, as applicable, and which IPT was being tested. These linear regressions enabled us to report on each factor (e.g., the impact of age or the impact of being assigned to the **examples** condition) while controlling for the other factors. Because we wanted to investigate the extent to which each of these factors (IVs) impacted the success of IPT communication, we chose not to perform model selection, instead reporting the coefficients and p-values for all IVs.

Beyond these main linear regression models, we wanted to examine specific sub-questions (e.g., comparing receivers' performance based on specific characteristics of the IPT description they saw). For these one-off comparisons between two groups, we used Mann-Whitney U tests (abbreviated **MWU** when reporting p-values) as the number of test cases answered correctly does not follow a normal distribution, making the more common t-test inappropriate. For all statistical tests, we set $\alpha = .05$, though we also report potentially relevant results above this threshold as appropriate.

Note that senders assigned to the condition where they were instructed to send examples did not always include examples for every IPT description. We marked a pair to have **consistently sent examples** if at least two of their four IPT descriptions included examples. We chose this threshold after observing negligible differences in average performance relative to the 4/4 case.

We also performed qualitative analysis to assess the characteristics of participants' procedures, examples, and interactions. Two researchers inductively developed a code book following a thematic analysis approach. Procedures, examples, and interactions all had separate code books, although they were all developed inductively. The researchers regularly met to discuss if new codes needed to be added. After coding, the researchers met and resolved all differences between their respective codes. We grouped codes that had less than five occurrences with thematically similar codes.

3.8 Limitations

While we made strong efforts to ensure that our study methods are robust, they still suffer from some limitations. In particular, while we made good-faith efforts to ensure that our selected IPTs were broadly representative of common programming tasks, it is ultimately impossible to claim that they are representative of programming problems as a whole due to their limited number. Similarly, while we study this subset of IPTs to inform the feasibility of end-user programming with LLMs, there is no guarantee that end-users will choose programming workloads consistent with the IPTs we chose to study. In this vein, we note that we phrased IPTs specifically to study IPT communication. Namely, we couch IPTs in non-programming-specific language, and include superfluous information to force senders to meaningfully engage with them during IPT communication, rather than simply parroting the phrasing we provided. This may not be representative of how non-programmers will engage in communicating programming tasks to LLMs in general. Finally, we note that the LLMs we query may have been trained on similar IPTs due to the common nature of some of our problems, such as finding the maximum element of a list. Individuals who query LLMs on less-common tasks may observe worse performance than what we report here.

CHAPTER 4
RESULTS

We now describe the results of our user study and LLM experiments.

4.1 Participants

Initially, 258 participants completed our study. However, we found evidence that a few participants may have used ChatGPT to rephrase IPTs. Two researchers independently coded responses for ChatGPT usage based on indicators like beginning an IPT with "Certainly! Here's a simplified algorithm procedure" or ending it with "I hope this is more general and easier to understand," both hallmarks of ChatGPT, as well as idiosyncratic behaviors, like listing all multiples of 11 and 13 from 0 to 1000 without being asked. Based on consensus from the two researchers, we discarded the 8 pairs who seemed to use ChatGPT, leaving 242 individuals as our data set.

Among these 242 participants, 10.3% did not have any college education, 34.3% had some college or an associate's degree, 37.2% of participants held a bachelor's degree, and 18.2% held a graduate degree. In terms of age, 12.4% of participants were 18–24 years old, 35.5% were 25–34 years old, 27.3% were 35–44 years old, and the remaining 9.5% were 45 years old or older. Finally, 52.5% of participants identified as male, 42.6% identified as female, and a total of 5.0% identified as non-binary/third gender, self-described, or preferred not to say. For our regression models, we merged smaller categories and always used the largest category as the baseline.

While we recruited our sample such that half of the participants reported prior programming experience and half did not on Prolific's platform filters (Section 3.1), a number of self-reported programmers fared poorly on the standardized programming quiz (Danilova et al. [2021]) we administered (Section 3.5). Based on both this quiz and self-reported experience, we categorized 72/242 participants (29.8%) as programmers and the rest as non-programmers.

4.2 Overall Correctness

Recall that each pair was randomly assigned four IPTs with four test cases each. Across their assigned IPTs, participants averaged 9.4/16 test cases correct (58.9%), with a median of 10/16 (62.5%) and standard deviation of 4.4. However, this average does not account for the participant's role, type of test case, or experimental condition. As we detail in the coming sections, each factor impacts correctness.

4.2.1 Senders vs. Receivers

Senders, who read the verbose descriptions we provided, answered more test cases correctly than receivers, who read their corresponding sender's description that rephrased, shortened, and generalized the verbose description. As Figure 4.1 illustrates, senders answered an average of 10.6/16 test cases correctly (66%), while receivers answered an average of 8.3/16 correctly (52%). This difference was significant (MWU, $p < .001$).

4.2.2 Regular vs. Edge Cases

Furthermore, participants did slightly better on regular test cases than edge cases, as shown in Figure 4.2. This difference was also statistically significant (MWU, $p = .030$).

4.2.3 Performance By IPT

Participants' performance varied substantially across the 12 IPTs we tested, which we chose to span a range of difficulty. Figure 4.3 displays the performance per IPT; recall that each had four test cases. Participants answered the most test cases correctly for `max` (mean=3.1/4,

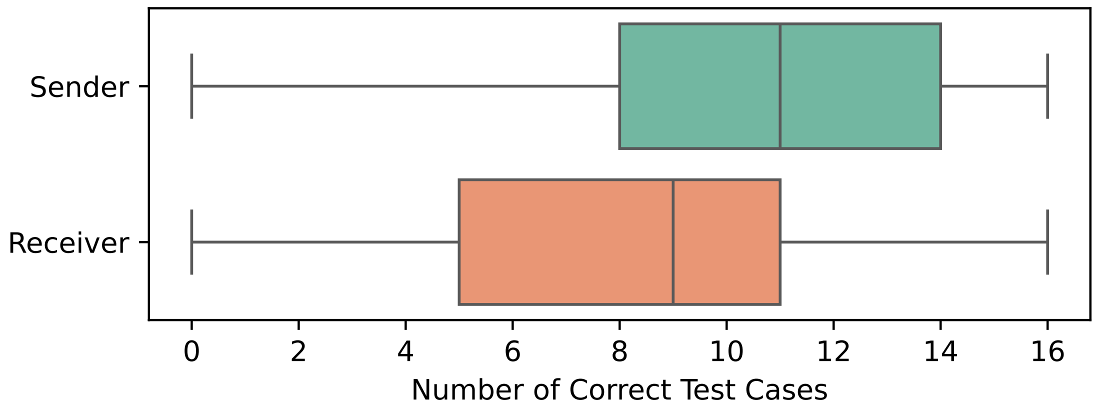


Figure 4.1: The distribution of how many of the 16 test cases each sender and receiver answered correctly.

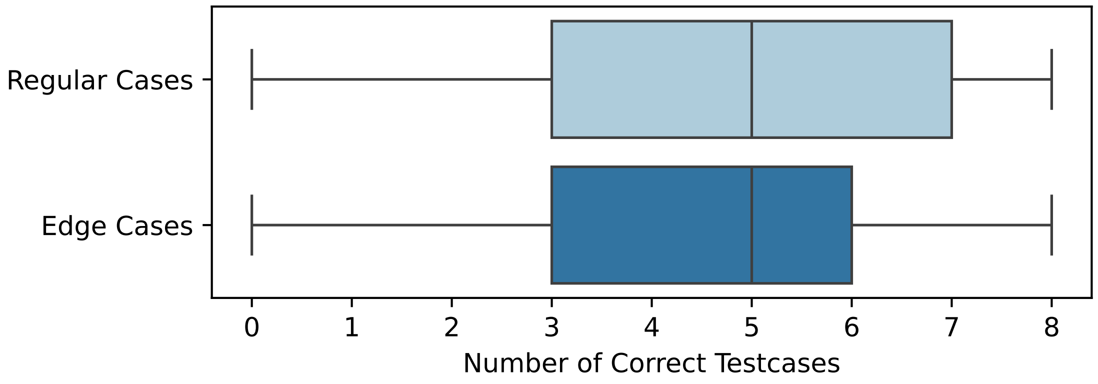


Figure 4.2: The distribution of how many of the 8 regular cases and 8 edge cases each participant answered correctly.

median=4) and the least for `fizzbuzz` (mean=1.1, median=1.0). In general, participants did well at answering test cases for prose descriptions of straightforward mathematical, string, and list operations, such as identifying palindromes, summing only the positive numbers in a list, counting the number of even numbers, or solving a variant of the subset sum problem. Given that half of the participants were reading the sender's description written by another participant in the study, this high performance answering test cases for such IPTs also indicates that most senders successfully communicated IPTs in prose, which is necessary for LLM-based interfaces for end-user programming.

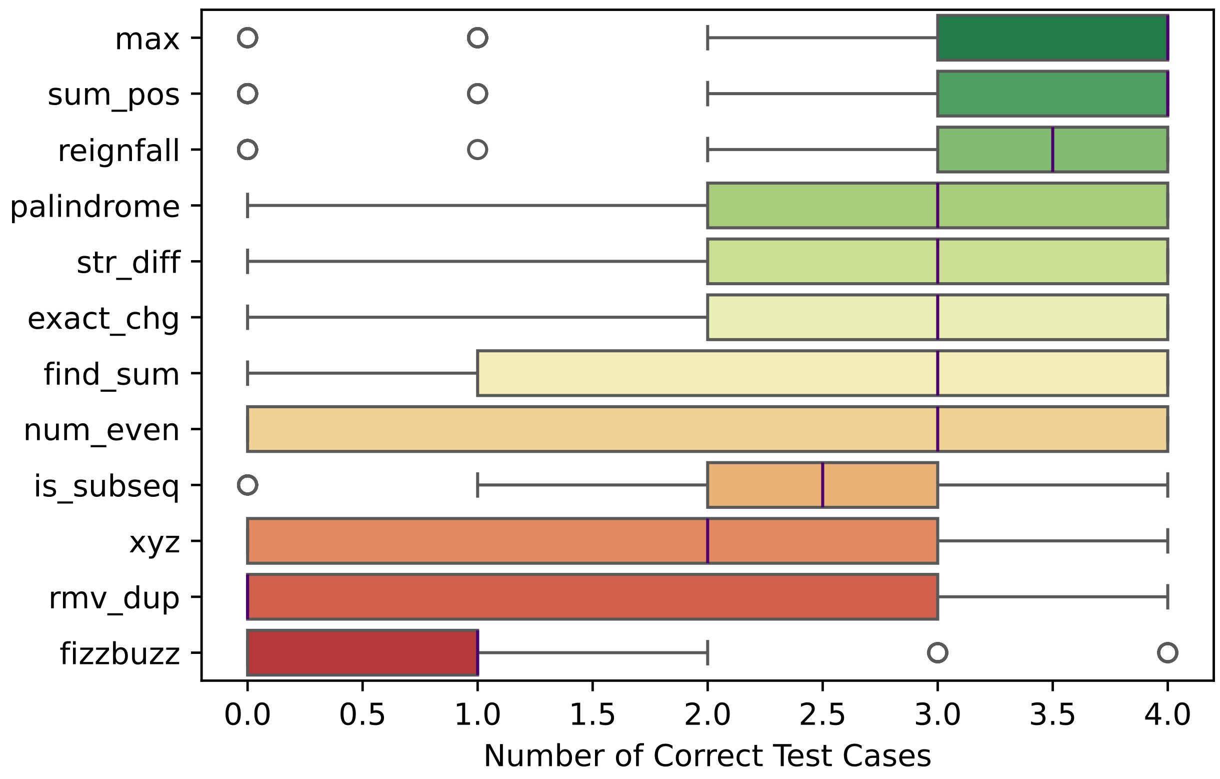


Figure 4.3: The distribution of how many of the 4 test cases for each IPT participants answered correctly.

That said, participants tended to do worse for more complex problems, notably `xyz` (which involved defining a non-trivial pair of expressions) and `fizzbuzz`. Both required multi-step mathematical calculations. However, `rmv_dup` also proved challenging for participants, despite its lack of computational complexity. This task involved removing duplicates from a list. In participants' free-text responses, we observed trends stemming from the ambiguities of natural language. That is, "remove duplicates" may require one to remove *all* occurrences of duplicates, or to keep the first occurrence but remove subsequent occurrences. While we precisely worded our verbose description to specify the former, many participants' natural language descriptions were less precise. We observed a similar failure mode for `num_even`, where many unsuccessful participants returned the *set of even numbers* in the list, rather than the *count of the even numbers* in the list.

4.2.4 The Contents of Senders' Descriptions

To determine where unsuccessful descriptions might have gone wrong, we qualitatively coded each pair's final IPTs (i.e., reflecting any clarification in the interactive condition). Our codes assessed the degree to which senders' descriptions generalized IPTs, as well whether they included all information we deemed necessary to answer the test cases. We omit 8 senders' descriptions that were barely rephrased from the verbose descriptions we gave, leaving 476 descriptions.

Figure 4.4 summarizes the characteristics of senders' descriptions. Senders usually included the information needed to answer the test cases. Specifically, 246 of the senders' descriptions (51.7%) included all information we deemed necessary, 152 (31.9%) included some of this information, and only 72 (15.1%) included none of it. However, we found mixed success in *generalization*, our instruction to senders to rewrite the context-specific verbose descriptions (e.g., about keeping score in a sports game) to general form (e.g., about list operations). While 163 senders' descriptions (34.2%) followed instructions and were fully generalized, and 76 (16.0%) were partially generalized, the remaining 231 (48.5%) were not generalized.

Figure 4.5 shows how these two characteristics of senders' descriptions corresponded to the receivers' performance answer test cases. Naturally, senders' descriptions that included more information relevant to the IPT correlated with better receiver performance. Specifically, senders' descriptions that included *all* relevant information led to receivers answering 2.7 of the 4 test cases on average (67.5%). Senders' descriptions that included only *some* of the information led to receivers answering 1.6 of the 4 test cases on average (40.6%), while those that included *no information* led to receivers answering 0.8 of the 4 test cases on average (20.2%). While the generalization of the senders' description had much less of an impact on receivers' success, we did observe that receivers tended to do better when the IPT was either fully generalized or not generalized at all, rather than partially generalized.

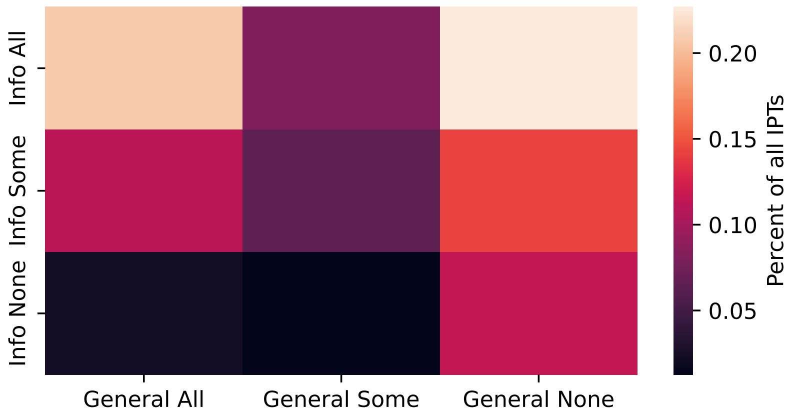


Figure 4.4: A heat map showing the degree to which senders' descriptions generalized the IPT alongside whether we felt they included all information needed to solve the test cases.

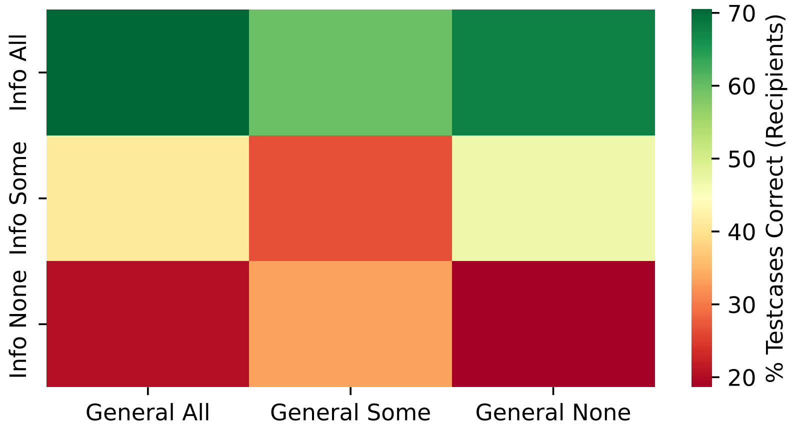


Figure 4.5: A heat map showing the fraction of test cases receivers answered correctly based on the characteristics of the sender's description identified in our qualitative coding.

Specifically, *full* generalization led to receivers correctly answering 2.3 test cases on average (57.5%), *some* generalization led to receivers correctly answering 1.8 test cases on average (44.0%), and *no* generalization led to receivers answering 2.0 test cases on average (49.8%).

4.3 Regressions

In the remainder of this section, we report the results of a series of linear regression models (see Section 3.7). For each, the DV is how many of the four test cases were answered correctly

Table 4.1: Linear regression whose dependent variable indicates how many of the four test cases *senders* answered correctly based on our verbose descriptions. $R^2 = 0.346$.

Independent Variable	**Baseline**	β	**SE**	**t**	**p**
(Intercept)	–	3.115	0.237	13.136	<**.001**
Sender is Programmer	Non-programmer	0.412	0.141	2.934	**.004**
Sender Age 18–24	25–34	0.038	0.207	0.183	.855
Sender Age 35–44	25–34	-0.015	0.144	-0.107	.915
Sender Age 44+	25–34	-0.695	0.167	-4.163	<**.001**
Sender is Non-male	Male	-0.017	0.125	-0.135	.893
Sender Degree: None	Bachelor's	-0.018	0.141	-0.125	.901
Sender Degree: Associate's	Bachelor's	0.044	0.212	0.209	.835
Sender Degree: Graduate	Bachelor's	0.270	0.173	1.557	.120
Condition: Examples	Non-examples	0.434	0.155	2.809	**.005**
Examples Sent	None Sent	0.325	0.187	1.738	.083
Condition: Interactive	Non-interactive	0.122	0.129	0.942	.346
Participants Interacted	Did Not	-0.523	0.234	-2.234	**.026**
Interaction Coded As Useful	Was Not	0.678	0.311	2.179	**.030**
IPT: `num_even`	`max`	-0.840	0.277	-3.035	**.003**
IPT: `sum_pos`	`max`	0.199	0.270	0.737	.461
IPT: `reignfall`	`max`	0.060	0.294	0.203	.839
IPT: `palindrome`	`max`	-0.386	0.267	-1.447	.149
IPT: `find_sum`	`max`	-0.786	0.277	-2.837	**.005**
IPT: `rmv_dup`	`max`	-1.931	0.264	-7.325	<**.001**
IPT: `str_diff`	`max`	-0.590	0.266	-2.217	.027
IPT: `is_subseq`	`max`	-0.657	0.271	-2.421	.016
IPT: `exact_chg`	`max`	-0.491	0.262	-1.876	.061
IPT: `fizzbuzz`	`max`	-2.211	0.268	-8.255	<**.001**
IPT: `xyz`	`max`	-1.231	0.278	-4.432	<**.001**

for a given IPT. When presenting the IVs, such as the relevant participant's demographics and assigned condition, we report β (the coefficient) and the p-value for that IV. For categorical variables, as most of our IVs are, β indicates how many more of the four test cases on average were answered correctly relative to the baseline category.

Table 4.1 presents our regression results for senders answering test cases based on our verbose descriptions, while Table 4.2 presents the corresponding regression results for receivers answering test cases based on their paired sender's description. Both regressions include IVs for the assigned conditions, the IPT, and the demographics of the participant answering test cases (respectively, the sender and the receiver). To better understand how the characteristics of the person writing the description impacted another person's ability to understand the IPT, the regression for receivers also included IVs for the demographics of the sender.

Echoing Section 4.2.3, we found that both senders and receivers answered fewer test cases correctly for four of the IPTs—`num_even`, `rmv_dup`, `fizzbuzz`, and `xyz`—compared to our regression baseline, `max`. Additionally, senders (but not receivers) also answered fewer test cases correctly for `find_sum` than for `max`.

Controlling for all other factors represented by IVs, we found that the number of test cases the sender answered correctly was significantly correlated with the number of test cases the receiver answered correctly ($\beta = 0.294$, $p < .001$). Intuitively, this factor indicates that senders who better understood the IPT as presented in our verbose description were better able to rephrase and communicate that IPT to the receiver.

We also analyzed demographic correlations with performance. For both senders and receivers, we found that the gender of the participant was *not* significantly correlated with performance answering test cases. Compared to our baseline (largest) age category, 25–34, both senders ($\beta = -0.695$, $p < .001$) and receivers ($\beta = -0.347$, $p = .041$) age 44+ answered fewer test cases correctly, as did receivers age 18–22 ($\beta = -0.519$, $p = .017$). Furthermore, receivers who were given descriptions written by senders age 18–24 answered fewer test cases correctly than receivers who were given descriptions written by senders in our baseline group, those age 25–34 ($\beta = -0.734$, $p = .003$). For the most part, education level was not a significant factor. The only exception was that receivers who were given a description written by a sender with a graduate degree answered more test cases correctly ($\beta = 0.410$, $p = .033$) than those given a description written by a sender with a bachelor's degree (our baseline). Appendix A.5 plots these demographic correlations.

4.4 Programmers vs. Non-Programmers (RQ 1)

RQ 1 asked what role prior programming experience plays in successful IPT communication. We found that senders with programming experience answered on average 0.412 more of the 4 test cases per IPT correctly than senders without programming experience ($\beta = 0.412$, $p = .004$). While this effect was statistically significant, it was not completely overwhelming (representing on average less than half a test case difference), highlighting that both programmers and non-programmers were able to answer test cases based on our verbose natural language descriptions.

We observed a slightly different effect for receivers. Receivers with programming experience answered on average 0.218 more of the 4 test cases per IPT correctly than receivers without programming experience, and this effect was not statistically significant ($\beta = 0.218$, $p = .136$). What mattered somewhat more was whether the sender who wrote the description for the receiver had programming experience. Receivers whose corresponding sender had programming experience answered on average 0.387 more of the 4 test cases per IPT correctly than receivers whose corresponding sender did not ($\beta = 0.387$, $p = .013$).

In short, we conclude that it is feasible for non-programmers to both describe IPTs in natural language and interpret others' natural language descriptions of IPTs with only slightly worse performance (roughly an 8.1% absolute difference overall) compared to programmers. Figure 4.6 plots these phenomena per participant (summing test cases across all 4 IPTs each saw, leading to a maximum of 16).

However, observing the difference in distributions is insufficient to understand *why* programmers do better. To fully answer RQ 1b, "what do programmers do that end-users do not do," we look at both examples (Section 4.5) and interaction patterns (Section 4.6).

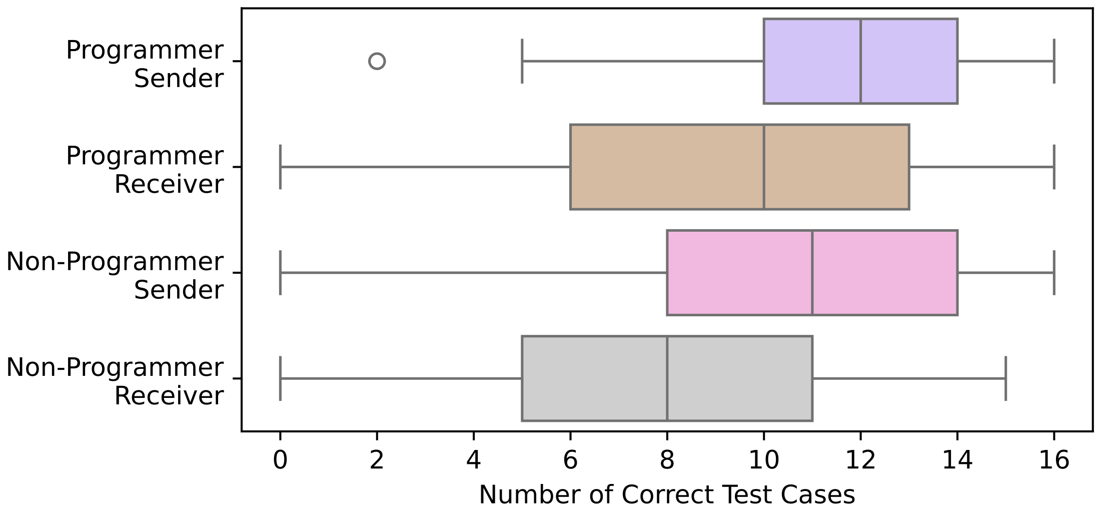


Figure 4.6: The distribution of how many of the 16 test cases (across IPTs) each participant answered correctly based on role and programming experience.

4.5 The Role of Examples (RQ 2)

RQ 2 asked what role the inclusion of example test cases (sample pairs of inputs and corresponding outputs) plays in successful communication. Senders who were shown examples that we provided alongside their verbose descriptions answered on average 0.434 more of the 4 test cases per IPT correctly than senders not shown examples ($\beta = 0.434$, $p = .005$). Whether or not that sender subsequently chose to send examples to their corresponding receiver also correlated weakly (non-significantly) with how many test cases the *sender themselves* answered correctly ($\beta = 0.325$, $p = .083$).

Whether the sender chose to send examples to their corresponding receiver correlated more strongly, however, with how many test cases their corresponding *receiver* answered correctly ($\beta = 0.506$, $p = .012$). Figure 4.7 plots these phenomena, summing across all four IPTs each participant saw. Notably, however, only 32 of the 48 senders randomly assigned to our examples condition (in which our instructions asked them to provide examples to the receiver), or 66.7%, did so. As such, future interfaces might consider automatically detecting whether a natural language description includes examples. In any case, the inclusion of

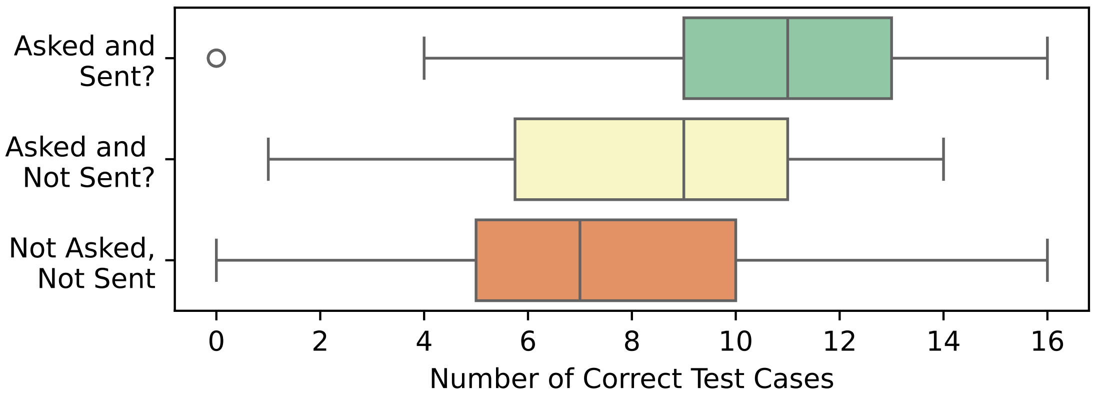


Figure 4.7: The distribution of receiver performance based on whether the sender included examples. A pair was considered to have sent examples if they did so for at least 2 of their 4 assigned IPTs. Only one pair sent examples when they were not asked to; for visual clarity, they are not shown.

examples alongside a natural language description of an IPT strongly correlates with better IPT communication (roughly an 18.1% absolute difference).

4.5.1 What types of examples were sent?

RQ 2a asked what types of examples are beneficial to IPT communication. To this end, we qualitatively coded the contents of examples participants provided, as well as their format. We observed four emergent themes. First, we applied the *format* code to examples formatted in a way that clearly define inputs and outputs, such as "Example: ['ghj', 'gh']; Report 'T'." Our *reuse* code indicated that at least one of the examples the sender provided the receiver was originally given by us to the sender and reused verbatim. We also coded sets of examples *all correct*, indicating that all provided input/output pairs are correct, because we noticed that some senders sometimes provided incorrect examples. Finally, our *explanation* code noted whether the examples included an explanation of why the output matched the input, such as mentioning "just add together the positive numbers to get the total sum, and ignore the negative points in the equation."

We used these codes to answer RQ 2a quantitatively. We found that senders' descriptions that contained examples that were entirely original, rather than copied from ours, resulted in the receiver answering more test cases correctly (MWU, $p = 0.030$). Similarly, descriptions whose examples were all correct, rather than including incorrect test cases, also resulted in the receiver answering more test cases correctly (MWU, $p = 0.008$). Figure 4.8 visualizes how these codes correlated with receivers' correctness.

4.5.2 Do senders with programming experience use different types of examples?

RQ 2b asked about differences in the types of examples senders included based on whether the sender had prior programming experience. First, programmers were more likely overall to send examples when assigned to the relevant condition. Specifically, programmers sent examples when their condition instructed them to do so 57% of the time (39/68 IPTs), while non-programmers sent examples only 38% of the time (56/148 IPTs). Programmers were 7% more likely to send original examples, 5% less likely to include explanations with their examples, and (surprisingly, given programming's precise nature) 8% less likely to format their examples.

4.6 The Impact of Interactivity (RQ 3)

RQ 3 asked whether interactions between the receiver and sender, such as asking for clarification, improved IPT communication. While we hypothesized that it would, we did not observe this to be the case. In this section, we considered whether participants were assigned to the condition where they could interact, as well as whether they actually interacted during the survey.

Interaction was somewhat rare. While 64 of the 131 pairs were assigned to the interactive condition, only 47 of those 64 pairs (73.4%) ever chose to interact. Furthermore, interaction did not appear to improve outcomes for the receiver; none of the relevant regression terms were significant. Surprisingly, we instead observed interaction to have a more substantial impact on senders. While pairs who interacted were associated with senders answering *fewer* test cases correctly ($\beta = -0.523$, $p = .026$), this effect was completely offset by that interaction being what we qualitatively coded as a useful interaction ($\beta = 0.678$, $p = .030$). In other words, senders whose corresponding receiver asked for clarification tended to perform slightly better at answering their own test cases only if their interaction meaningfully exchanged information about the IPT. Otherwise, they tended to perform worse, which we hypothesize to be because the senders' description may have revealed to the receiver a lack of understanding on the part of the sender.

4.6.1 What interaction outcomes were beneficial?

Because we observed interaction (either permitted by the condition or actually taken advantage of) to have minimal impact on the receiver's ability to answer test cases, we qualitatively analyzed the interactions that did occur. Many of them did not seem to clarify the IPT. In response, two researchers inductively developed a definition for useful interaction and repeatedly met to refine this definition until they reached high levels of agreement (Cohens $\kappa = 0.772$). They then qualitatively coded all procedures as useful or not, including reasons why they were or were not useful. The researchers then met and jointly resolved all differences.

Their ultimate definition of a **useful interaction** is one in which at least one of the following happens: the sender makes the *IPT structure clearer* in response (e.g.,, by adding headers to paragraphs); the sender responds with *new, relevant information*; the sender and receiver correctly *clarify the input and output* for the IPT; or they discuss the IPT in a way that *clarifies ambiguities.*

4.6.2 What clarification requests led to good outcomes?

Receivers' requests for clarification tended towards five main themes. Of the 84 requests, 14 (17.1%) *clarified input/output* pairs given the IPT phrasing, such as the outcome when given a certain edge case (8/14). Another 27 requests (33.0%) asked about the characteristics of certain *variables.* For example, for the IPT "report how many even numbers are in each number sequence," the receiver asked, "Even in terms of number count? ie. [1223] = one because there are two 2's. Or even in terms of the number itself? Ie. [1223] = two because there are two 2's."

Other requests for clarification were less likely to lead to useful outcomes. Another 9 requests (11.0%) asked about the *IPT's framing*, rather than information pertinent to the computational task itself. An example for the `find_sum` task, framed as finding the right lengths of scrap wood to make a door, was, "Do I have enough wood glue for such a project?" Furthermore, 19 (23.2%) broadly reported that the IPT was *unintelligible*, or that they were not clear on what they should report or how they should do so. For example, "It seems very wordy and confusing. I still don't know what two numbers in super to find or how i am supposed to do that." Finally, 13 requests (15.9%) did not neatly fall under the other categories, such as grammatical edits, or confusion over math symbols (e.g., confirming that > means "greater than").

Figure 4.9 visually summarizes how these requests mapped to outcomes. We concluded that requests to clarify variables and I/O were the most useful when describing an IPT in English prose. These requests led to positive outcomes 48.1% and 61.5% of the time, respectively. Pairs with at least one programmer were 12.2% less likely to interact than pairs with no programmers. Namely, pairs with one programmer interacted 65.6% of the time, while pairs with no programmers interacted 77.8% of the time.

4.7 Communication to LLMs (RQ 4)

Because our study's ultimate implications center on the ability for humans to communicate IPTs to LLM-based chatbots, rather than human receivers, our final set of analyses replaced the human receivers with a variety of modern LLMs. In other words, we fed the same IPT descriptions written by senders in our study to GPT-4, Gemini, and two variants each of Chat Llama and Code Llama. As described in Section 3.6, we tested both having the LLM directly answer the same test cases as the human receiver, as well as having it produce code that would then be used to answer the test cases.

Table 4.3 compares the performance of these LLMs against each other and against the human receivers from our study. We observe that, of the tested LLMs, GPT-4 consistently outperformed the other LLMs, so we primarily report on its behavior. We found that all LLMs except Gemini were more adept at answering test cases directly, rather than producing code to answer the test cases.

Furthermore, GPT-4 slightly outperformed human receivers, answering 59.8% of all test cases correctly. The corresponding human receivers answered 58.9% correctly (absolute difference <1%). However, this provides only a partial look at the LLMs' abilities. We now examine them through the lens of functional correctness, the de-facto standard for evaluating code.

Recall that functional correctness deems code to be correct if it successfully passes *all* associated test cases. In other words, human participants display functional correctness if they answer all four of an IPT's test cases correctly–in our data, this happens 34.7% of the time. In contrast, GPT-4 displayed functional correctness *28.9%* of the time when directly answering test cases, or 36.0% of the time when producing code to answer test cases. We observed that GPT-4 produced functionally correct code (per-IPT) 40.2% of the time when given rephrased IPTs from senders with programming experience, as opposed to 34.4% for senders without such experience.

Parallel to our regression models for humans, we also built regression models for the LLMs. Tables 4.4–4.5 in the body of the paper report those models for GPT-4, respectively: the cases where GPT-4 answered test cases directly (Table 4.4) and where GPT-4 wrote code (Table 4.5). The appendix contains parallel tables for Code Llama (Tables A.2–A.3) and Gemini (Tables A.4–A.5). As with human receivers, we found that the inclusion of examples in the sender's IPT description was associated with better performance by the LLM (in all six of these cases). Furthermore, in 4/6 cases, the sender being a programmer was associated with better performance by the LLM.

We further observed that GPT-4 was much more likely to produce code that was functionally correct when IPT communication was perfect (i.e., the corresponding human receiver answered all four test cases correctly), as shown in Figure 4.10. Similarly, if the IPT communication was *not* perfect (the corresponding human receiver did not answer 4/4 test cases correctly), then we observed a 73.3% chance that the LLM would produce incorrect code.

4.8 Perceptions

We asked participants about their perceptions of confidence and comprehension in our surveys. We observed a strong correlation between confidence in comprehension during the survey and performance on test cases, both per IPT ($\rho = 0.41$) and overall ($\rho = 0.45$). We found that programmers were typically better at predicting their performance on test cases than non-programmers.

We also asked participants about their opinions of, and experiences with, LLMs. Programmers were more likely to have seen and used GitHub Copilot, but both programmers and non-programmers had seen and used ChatGPT at similar rates. Both populations were significantly more familiar with ChatGPT than GitHub Copilot. In addition, both groups believed that LLMs could perform at least as well as humans, if not better, on the assigned tasks. However, while programmers almost always demonstrated more faith in LLMs' abilities than non-programmers, they were less likely to believe LLMs were capable of clear communication.

Table 4.2: Linear regression whose dependent variable indicates how many of the four test cases *receivers* answered correctly based on the sender's description. We included independent variables about both the receiver (who answered test cases) and the sender (who wrote the description). $R^2 = 0.340$.

Independent Variable	**Baseline**	β	**SE**	**t**	**p**
(Intercept)	–	1.731	0.342	5.054	<**.001**
Receiver is Programmer	Non-programmer	0.218	0.146	1.493	.136
Receiver Age 18–24	25–34	-0.519	0.216	-2.403	**.017**
Receiver Age 35–44	25–34	-0.260	0.186	-1.395	.164
Receiver Age 44+	25–34	-0.347	0.169	-2.052	**.041**
Receiver is Non-male	Male	0.165	0.130	1.268	.205
Receiver Degree: None	Bachelor's	-0.021	0.170	-0.122	.903
Receiver Degree: Associate's	Bachelor's	0.133	0.209	0.638	.524
Receiver Degree: Graduate	Bachelor's	-0.028	0.186	-1.515	.130
Condition: Examples	Non-examples	-0.000	0.173	0.000	.999
Examples Sent	None Sent	0.506	0.202	2.511	**.012**
Condition: Interactive	Non-interactive	0.057	0.140	0.406	.685
Participants Interacted	Did Not	-0.157	0.250	-0.629	.530
Interaction Coded As Useful	Was Not	0.037	0.333	0.112	.911
IPT: `num_even`	`max`	-0.695	0.297	-2.339	**.020**
IPT: `sum_pos`	`max`	0.023	0.287	0.080	.936
IPT: `reignfall`	`max`	-0.072	0.312	-0.230	.818
IPT: `palindrome`	`max`	-0.169	0.284	-0.597	.551
IPT: `find_sum`	`max`	-0.571	0.296	-1.927	.055
IPT: `rmv_dup`	`max`	-1.033	0.296	-3.495	<**.001**
IPT: `str_diff`	`max`	-0.103	0.284	-0.363	.717
IPT: `is_subseq`	`max`	-0.249	0.290	-0.859	.391
IPT: `exact_chg`	`max`	-0.175	0.278	-0.630	.529
IPT: `fizzbuzz`	`max`	-1.041	0.307	-3.391	<**.001**
IPT: `xyz`	`max`	-1.133	0.301	-3.771	<**.001**
Sender # Test Cases Correct	–	0.294	0.050	5.841	<**.001**
Sender is Programmer	Non-programmer	0.387	0.156	2.482	**.013**
Sender Age 18–24	25–34	-0.734	0.242	-3.034	**.003**
Sender Age 35–44	25–34	-0.196	0.157	-1.248	.213
Sender Age 44+	25–34	-0.203	0.194	-1.048	.295
Sender is Non-male	Male	-0.237	0.138	-1.713	.087
Sender Degree: None	Bachelor's	0.221	0.159	1.391	.165
Sender Degree: Associate's	Bachelor's	0.172	0.229	0.751	.453
Sender Degree: Graduate	Bachelor's	0.410	0.192	2.133	**.033**

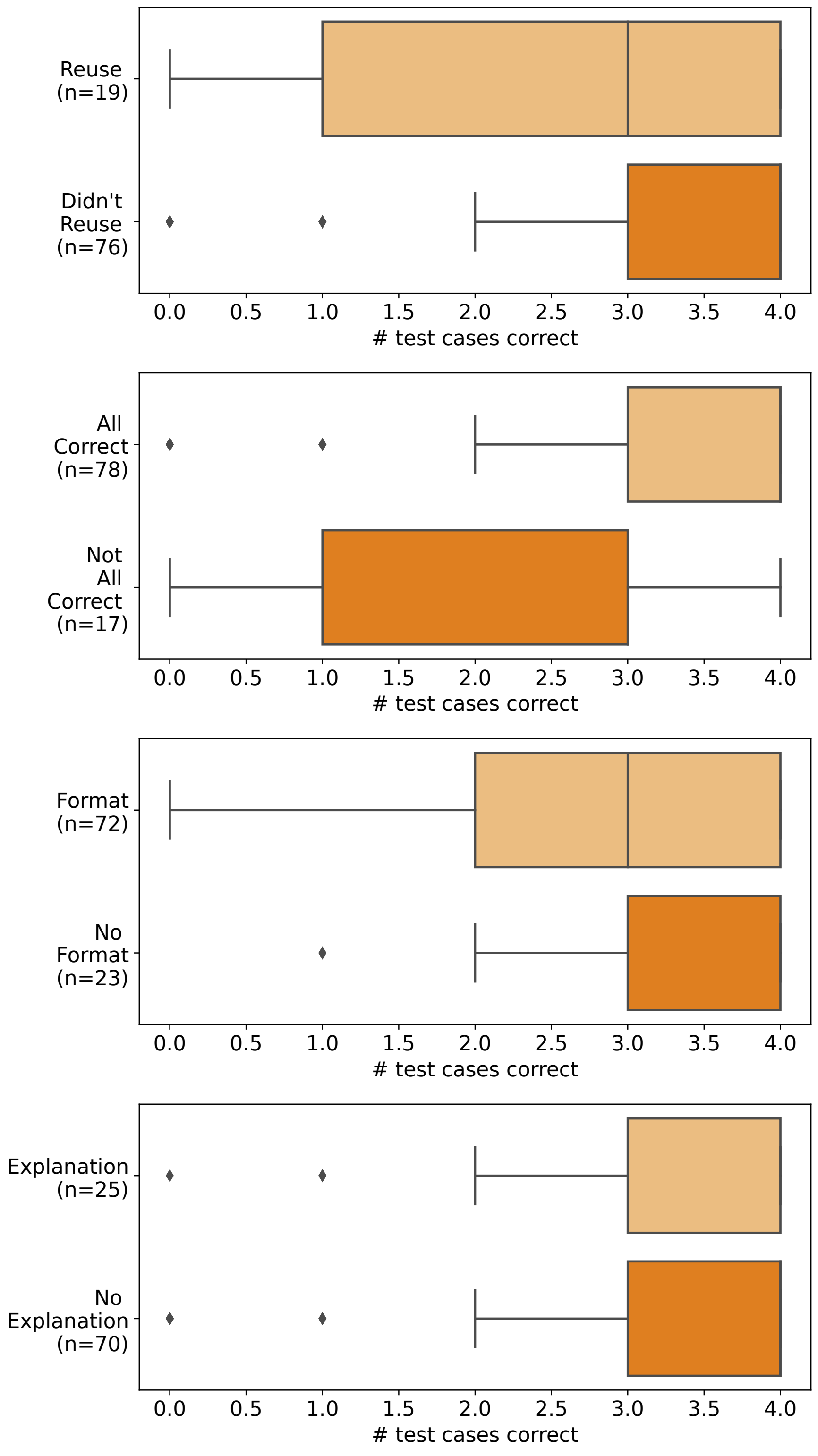


Figure 4.8: The distribution of how many test cases the receiver answered correctly for a given IPT based on qualitatively coded characteristics of senders' descriptions.

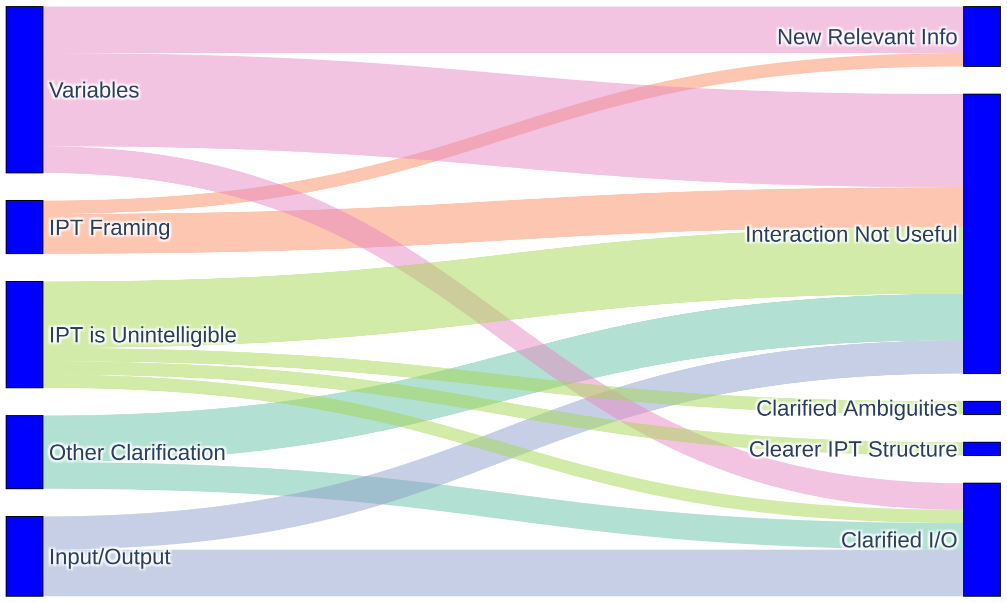


Figure 4.9: A sankey diagram connecting themes of receivers' clarification requests about senders' IPT descriptions and their associated outcomes. For visual clarity, we dropped all request-outcome pairs that only occurred once.

Table 4.3: Average performance of all tested LLMs and humans. "Direct Answering" indicates that the LLM directly responded to the testcase, while "Code" indicates that the LLM produced a Python function which was then run on testcases.

	Directly Answering		*Writing Code*	
Receiver	**Test Cases**	**All Correct**	**Test Cases**	**All Correct**
Human Participants	58.9%	25.6%	-	-
ChatLLaMA 7B	18.7%	2.9%	12.0%	6.0%
ChatLLaMA 13B	27.4%	4.3%	26.8%	17.1%
Code Llama 7B	33.4%	6.0%	32.8%	21.5%
Code Llama 34B	40.9%	9.3%	34.4%	22.5%
Gemini	27.1%	6.8%	38.3%	27.3%
GPT-4	59.8%	28.9%	45.6%	36.0%

Table 4.4: Linear regression with the dependent variable indicating how many of the four test cases *GPT-4* answered correctly when *directly answering* test cases. $R^2 = 0.267$.

Independent Variable	**Baseline**	β	**SE**	**t**	**p**
(Intercept)	–	2.701	0.249	10.860	<**.001**
Sender is Programmer	Non-programmer	0.255	0.147	1.732	.084
Sender Age 18–24	25–34	-0.052	0.217	-0.239	.812
Sender Age 35–44	25–34	-0.129	0.151	-0.854	.393
Sender Age 44+	25–34	-0.557	0.175	-3.179	**.002**
Sender is Non-male	Male	-0.138	0.132	-1.047	.296
Sender Degree: None	Bachelor's	0.084	0.148	0.570	.569
Sender Degree: Associate's	Bachelor's	0.132	0.222	0.597	.551
Sender Degree: Graduate	Bachelor's	0.152	0.182	0.833	.405
Condition: Examples	Non-examples	0.176	0.162	1.087	.277
Examples Sent	None Sent	0.403	0.196	2.052	**.041**
Condition: Interactive	Non-interactive	-0.031	0.136	-0.229	.819
Participants Interacted	Did Not	-0.539	0.245	-2.194	**.029**
Interaction Coded As Useful	Was Not	0.542	0.326	1.663	.097
IPT: `num_even`	`max`	-0.448	0.290	-1.542	.124
IPT: `sum_pos`	`max`	0.865	0.284	3.050	**.002**
IPT: `reignfall`	`max`	-0.142	0.308	-0.460	.646
IPT: `palindrome`	`max`	0.331	0.280	1.185	.236
IPT: `find_sum`	`max`	-0.293	0.290	-1.009	.313
IPT: `rmv_dup`	`max`	-1.429	0.276	-5.170	<**.001**
IPT: `str_diff`	`max`	-0.279	0.279	-1.001	.317
IPT: `is_subseq`	`max`	0.113	0.284	0.396	.692
IPT: `exact_chg`	`max`	-0.036	0.274	-0.131	.896
IPT: `fizzbuzz`	`max`	-1.488	0.281	-5.300	<**.001**
IPT: `xyz`	`max`	-1.038	0.291	-3.565	<**.001**

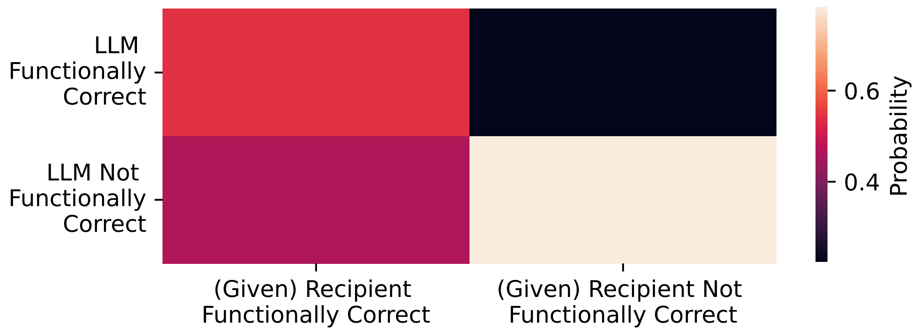


Figure 4.10: The probability of the LLM being functionally correct for an IPT (correctly answering all four test cases) relative to the functional correctness of the receiver's answers.

Table 4.5: Linear regression with the dependent variable indicating how many of the four test cases **GPT-4** answered correctly when *producing Python code*. $R^2 = 0.294$.

Independent Variable	**Baseline**	β	**SE**	**t**	**p**
(Intercept)	–	1.748	0.283	6.171	<**.001**
Sender is Programmer	Non-programmer	0.336	0.168	2.004	**.046**
Sender Age 18–24	25–34	0.209	0.247	0.846	.398
Sender Age 35–44	25–34	-0.102	0.172	-0.594	.553
Sender Age 44+	25–34	-0.928	0.200	-4.651	<**.001**
Sender is Non-male	Male	-0.225	0.150	-1.503	.133
Sender Degree: None	Bachelor's	0.076	0.168	0.452	.651
Sender Degree: Associate's	Bachelor's	0.215	0.253	0.850	.396
Sender Degree: Graduate	Bachelor's	0.387	0.207	1.868	.062
Condition: Examples	Non-examples	0.058	0.185	0.315	.753
Examples Sent	None Sent	0.630	0.224	2.818	**.005**
Condition: Interactive	Non-interactive	-0.030	0.155	-0.193	.847
Participants Interacted	Did Not	-0.314	0.280	-1.122	.262
Interaction Coded As Useful	Was Not	-0.017	0.372	-0.046	.963
IPT: `num_even`	`max`	1.033	0.331	3.122	**.002**
IPT: `sum_pos`	`max`	1.790	0.323	5.541	<**.001**
IPT: `reignfall`	`max`	0.842	0.351	2.398	**.017**
IPT: `palindrome`	`max`	1.687	0.319	5.295	<**.001**
IPT: `find_sum`	`max`	0.089	0.331	0.270	.788
IPT: `rmv_dup`	`max`	-0.771	0.315	-2.448	**.015**
IPT: `str_diff`	`max`	0.076	0.318	0.238	.812
IPT: `is_subseq`	`max`	0.159	0.324	0.489	.625
IPT: `exact_chg`	`max`	0.226	0.313	0.723	.470
IPT: `fizzbuzz`	`max`	-0.640	0.320	-2.001	**.046**
IPT: `xyz`	`max`	0.073	0.332	0.219	.827

CHAPTER 5
DISCUSSION

We ultimately observe that communication of IPTs is possible for both programmers and non-programmers. However, programmers are more likely to take actions that correlate with positive outcomes. We believe this is most clear when examining clarification requests. Clarification requests tended to have more positive outcomes when the requests concerned the values that variables could take, or otherwise specified the I/O of an IPT. We speculate that these have the most impact because they are specific enough to act upon (as opposed to simply reporting that the IPT is unintelligible), and because IPTs are evaluated for correctness based on whether their transformation of input to output is correct. Thus, any requests to clarify I/O are inherently more likely to result in more successful IPT communication. Part of programmers' success may be due to a natural conception of IPTs as input-output transformations.

We also note that examples play a key role in IPT communication between both humans and LLMs. We hypothesize that being able to formulate examples serves as an effective indicator of IPT comprehension, because it requires an individual to understand the essence of an IPT: namely, the IPT's transformation of input to output. We speculate that examples have so much impact because understanding the input-to-output transformation is critical to understanding the IPT itself. That is, input-output pairs may be the most concise and accurate way to specify an IPT. This would explain why they are so impactful, both for humans who have the capacity to reason about the nature of the I/O transformation itself, and for LLMs, which must simply probabilistically guess the most likely transformation during code production. Our results suggest that making example generation a core component in future end-user programming interfaces will encourage more accurate results.

Finally, we remark on the possibility of using LLMs as interfaces to enable end-user programming. We ultimately find that communication with LLMs is more or less successful based on similar criteria to effective human-to-human IPT communication: namely, the presence of examples, the sender having programming experience, and the presence or absence of useful interaction. We speculate that because we observe both programming experience and sender/recipient designation to have small effects in comparison to the presence or absence of examples, it is entirely possible for end users to program as effectively as those with programming experience by using LLMs. Further, we hypothesize that differences in performance caused by programming experience may be overcome through the development of interfaces that encourage the inclusion of examples during LLM prompting. Additionally, promoting certain *types* of clarification, such as those concerning I/O or function variables, during human-LLM coding may have strong implications for recovery from initially unsuccessful IPT communication.

CHAPTER 6

CONCLUSION

In this paper, we examine the communication of information processing tasks, or IPTs. We do so by examining the expected input and output of these IPTs, post-communication. In particular, we examine how humans communicate IPTs to other humans, as well as how they communicate them to code-generating LLMs. We find that a small amount of accuracy is lost during transmission, as human senders tend to do better at answering test-cases than human receivers (14% absolute difference). We further note a similar small difference in which human programmers tend to do better than end-users (8.1% absolute difference), and in particular, they are more likely to take certain actions that correlate with better IPT communication such as sending examples. We find that allowing for interaction during IPT communication enables participants who did a bad job rephrasing IPTs to fix their shortcomings, however, positive outcomes tend to correlate with requests to clarify IPT I/O. Finally, we note that LLMs seem to be sensitive to many of the same things as humans during IPT communication, particularly the presence of examples, sender programming experience, and the presence of useful interaction. We hypothesize that with the development of interfaces that encourage certain types of interaction and the sending of examples during IPT communication, end-users can program with LLMs as effectively as programmers.

APPENDIX A
ADDITIONAL DETAILS

A.1 Additional Details About Selecting IPTs

We wanted to ensure that our programming tasks were appropriately representative of problems that programmers would be likely to face. As a sanity check, we randomly sampled 50 questions each from LeetCode, MIT introductory programming courses, and HumanEval. We did not sample problems from APPS, MBPP, and BigBench because of either high problem complexity (as in the case of APPS) and/or high tendency to invoke mathematical or algorithmic definitions (as in the case of MBPP and BigBench). The set of n=150 questions was then randomly ordered. Two researchers inductively developed the codebook in Table A.1 to describe the types of problems, and qualitatively coded the questions. The researchers then met to resolve coding differences.

Table A.1: Primary attributes on which we categorized potential IPTs.

Category	**Description**
List Operations	The task involves list manipulation and/or working with a list to compute a value (e.g., find the minimum of a list of values). 1D arrays are lists.
String Operations	The task involves string manipulation and/or working with a list to compute a value (e.g., given a string, determine if something is a substring).
Simple Filtering	The task involves simple "filtering": given a number of values, find a specific value (e.g., find a max, remove values that meet a certain criteria).
Optimization	The task involves finding an optimal value (e.g., shortest path). This is distinct from filtering in that the sub-tasks required are nontrivial.
Two Dimensional	The task involves 2D arrays, and/or thinking in with a "grid" pattern (e.g., moving around obstacles in a grid, battleship).
Simple Arithmetic	The task involves simple arithmetic operations (addition, subtraction, division, modulo etc).
Complex Arithmetic	The task involves more complex arithmetic operations (computing derivatives, efficiently checking if a number is prime).
Examples	The task includes examples of intended input/output.
Data Structures	The task involves some manipulation of a data structure (e.g., a binary tree or linked list).
Boolean/Branching	The task revolves around using conditionals or other boolean logic.

Following that, the researchers qualitatively coded our selected 12 programming tasks using the codebook developed over the set of 150. The researchers similarly met to resolve coding differences. We then calculated the relative frequency of these labels among our randomly sampled 150 programming problems and 12 programming tasks, and compared the ratios against each other to ensure that the qualities of our chosen programming tasks were appropriate. We note that the task types varied significantly based on the problems source. We ultimately concluded that many common programming tasks may be too difficult for a novice to understand. In particular, N-dimensional arrays, data structure questions, and optimization problems were all common, but would be too difficult for an individual without programming experience to understand. We consequently chose our tasks to emphasize simple arithmetic and simple logic.

A.2 Survey Instrument

A.2.1 Pre-survey

The aim of this study is to examine how people communicate computational tasks, or structured processes that can be automated by a computer program. No programming experience is required to participate in this study. After completing the demographic questions on the next page, you will be directed to another website, where the main portion of the study will take place.

Please answer the following demographic questions.

1. How old are you? • Under 18 • 18-24 years old • 25-34 years old • 35-44 years old • 45-54 years old • 55-64 years old • 65+ years old
2. What is the highest level of education you have completed? • Some high school or less • High school diploma or GED • Some college, but no degree • Associates or technical degree • Bachelor's degree • Graduate or professional degree (MA, MS, MBA, PhD, JD, MD, DDS, etc.) • Prefer not to say

3. How do you describe yourself? • Male • Female • Non-binary / third gender • Prefer to self-describe • Prefer not to say

This next portion of the study will evaluate how people communicate programming tasks to each other. Here is some relevant terminology:

- A **procedure** will describe something to do. For example, a procedure might be "Pick the cheapest price in the list of food item prices."
- A **test case** is an input which you will be asked to compute an answer for, given a procedure. In the above example of a procedure, a test case is a list [1.00, 2.00, 0.50]. The correct answer to this test case is "0.50".

You will now be directed to another website. At this site, you will be paired with another participant. You will interpret four procedures of randomized difficulty and answer test cases. If the redirection fails to work, please return your submission, as this is indicative of a larger technical error caused by your browser configuration.

A.2.2 Post-survey

We will now ask you several follow-up questions regarding your experience participating in our study.

1. I feel that I understood what I was supposed to be doing in this study. • Strongly disagree • Disagree • Somewhat disagree • Neither agree nor disagree • Somewhat agree • Agree • Strongly Agree
2. *(If Strongly disagree, Disagree, Somewhat disagree selected)* If any part of the study process was hard to understand, please elaborate.

We will now ask questions about your experience relating to the procedures and test cases you saw during the study.

3. I felt that it was easy to answer test cases using the procedures • Strongly disagree • Disagree • Somewhat disagree • Neither agree nor disagree • Somewhat agree • Agree •

Strongly Agree

4. *(If Strongly disagree, Disagree, Somewhat disagree selected)* What did you find to be **difficult** about using the procedures provided to answer the test cases?
5. *(If Strongly agree, Agree, Somewhat agree selected)* What did you find to be **easy** about using the procedures provided to answer the test cases?
6. What information that you might have found useful, if any, was missing from the four procedures you were given?

This block of 5 questions was shown only to participants assigned to be a receiver

1. I believe that the procedures provided to me **initially, before the other participant made any clarifications I requested**, were clearly understandable. • Strongly disagree • Disagree • Somewhat disagree • Neither agree nor disagree • Somewhat agree • Agree • Strongly Agree
2. I believe that the procedures provided to me **after the other participant made any clarifications I requested**, were clearly understandable. • Strongly disagree • Disagree • Somewhat disagree • Neither agree nor disagree • Somewhat agree • Agree • Strongly Agree
3. I believe that the procedures provided to me were clearly understandable. • Strongly disagree • Disagree • Somewhat disagree • Neither agree nor disagree • Somewhat agree • Agree • Strongly Agree
4. I was sometimes confused by the four procedures the other participant wrote. • Strongly disagree • Disagree • Somewhat disagree • Neither agree nor disagree • Somewhat agree • Agree • Strongly Agree
5. *(If Strongly agree, Agree, Somewhat agree selected)* What was confusing about the other participant's procedures?

This block of 12 questions was shown only to participants assigned to be a sender

The next two questions aim to understand your approach to rephrasing the procedure.

1. What information did you make sure to include in your rephrasing of the procedures?
2. Please describe how you structured your rephrasing of the procedures.

These next questions relate more generally towards your experiences with the procedures and test cases you encountered during this study.

3. I believe that the procedures provided to me were clearly understandable. • Strongly disagree • Disagree • Somewhat disagree • Neither agree nor disagree • Somewhat agree • Agree • Strongly Agree
4. *(If Strongly disagree, Disagree, Somewhat disagree selected)* What about the procedures made them difficult to understand?
5. *(If Strongly agree, Agree, Somewhat agree selected)* What about the procedures made them easy to understand?
6. I believe that the rephrased procedures I sent the other participant were clearly understandable. • Strongly disagree • Disagree • Somewhat disagree • Neither agree nor disagree • Somewhat agree • Agree • Strongly Agree
7. I believe that the other participant **would have difficulty** answering test cases correctly using my rephrased procedures. • Strongly disagree • Disagree • Somewhat disagree • Neither agree nor disagree • Somewhat agree • Agree • Strongly Agree
8. What did you find to be **easy** about rephrasing the procedures?
9. What did you find to be **difficult** about rephrasing the procedures?
10. *(If assigned to the interactive condition)* Interacting (chatting, revising procedures) with the other participant improved my rephrased version of the procedure. • Strongly disagree • Disagree • Somewhat disagree • Neither agree nor disagree • Somewhat agree • Agree • Strongly Agree
11. *(If Strongly disagree, Disagree, Somewhat disagree selected)* Why did interacting with the other participant **not** help you improve your rephrasing of the procedure?
12. *(If Strongly agree, Agree, Somewhat agree selected)* Why did interacting with the other

participant help you improve your rephrasing of the procedure?

This block of 6 questions was shown only to participants assigned to the interactive condition

1. Interacting (chatting, revising procedures) with the other participant improved my understanding of the procedure. • Strongly disagree • Disagree • Somewhat disagree • Neither agree nor disagree • Somewhat agree • Agree • Strongly Agree
2. *(If Strongly disagree, Disagree, Somewhat disagree selected)* Why did interacting with the other participant not improve your understanding of the procedure?
3. *(If Strongly agree, Agree, Somewhat agree selected)* Why did interacting with the other participant improve your understanding of the procedure?
4. Interacting (chatting, revising procedures) with the other participant did not help me avoid mistakes • Strongly disagree • Disagree • Somewhat disagree • Neither agree nor disagree • Somewhat agree • Agree • Strongly Agree
5. *(If Strongly agree, Agree, Somewhat agree selected)* Why did interacting with the other participant help prevent you from making mistakes?
6. *(If Strongly disagree, Disagree, Somewhat disagree selected)* Why did interacting with the other participant **not** help prevent you from making mistakes?

The following questions are meant to test your knowledge of programming. If you've never programmed before, you likely will not know the answer; in that case, please make your best guess.

1. Which of these values would be most fitting for a Boolean? • Small • I don't know • Solid • Quadratic • Red • True
2.

```
main{
 print(func("hello world"))
}

String func(String in){
   int x = len(in)
   String out = ""
   for(int i = x-1; i >= 0; i--){
       out.append(in[i])
   }
   return out
}
```

What is the parameter of the function? • String out • String in • I don't know • int $i = x - 1; i >= 0; i - -$ • Outputting a String • int x = len(in)

3. What is your experience with computer programming? Please select as many or as few options that apply. • I have completed a course (in-person or online) focused on computer programming • I hold a degree in computer science, computer engineering, IT, or similar • Computer programming has been part of my responsibilities for a job • I have used computer programming for a hobby (i.e., non-academic, non-work) project • I don't have any of the experience listed above
4. Please list all programming languages you are proficient in. Write "none" if you are not proficient in any programming language.

In recent years, great strides have been made in advancing artificial intelligence (AI), and AI-powered chatbots are no exception. Some of these chatbots are capable of taking a description of a problem and providing answers based on that description, similar to the exchange you just experienced.

5. In this study, you were paired with another human participant. Would you have done anything differently, regarding any aspect of the study, if you were paired with an AI rather than a person?
6. Do you expect that an AI or a human would be better at clearly communicating procedures like those in this study? • Definitely an AI • Probably an AI • Possibly an AI • They

would be equal • Possibly a human • Probably a human • Definitely a human

7. Do you expect that an AI or a human would be better at understanding procedures like those in this study? • Definitely an AI • Probably an AI • Possibly an AI • They would be equal • Possibly a human • Probably a human • Definitely a human
8. Do you expect that an AI or a human would be better at correctly solving test cases like those in this study? • Definitely an AI • Probably an AI • Possibly an AI • They would be equal • Possibly a human • Probably a human • Definitely a human
9. *(Attention check)* For this question, please mark "They would be equal" • Definitely an AI • Probably an AI • Possibly an AI • They would be equal • Possibly a human • Probably a human • Definitely a human
10. Do you expect that an AI or a human would be better at writing procedures? • Definitely an AI • Probably an AI • Possibly an AI • They would be equal • Possibly a human • Probably a human • Definitely a human
11. How familiar are you with Github Copilot/Github Copilot-X? • Very unfamiliar • Moderately unfamiliar • Somewhat unfamiliar • Neither familiar nor unfamiliar • Somewhat familiar • Moderately familiar • Very familiar
12. Have you used Github Copilot//Github Copilot-X? • Yes • No
13. *(If Yes selected)* What was your experience using Copilot/Github Copilot-X like?
14. How familiar are you with ChatGPT? • Very unfamiliar • Moderately unfamiliar • Somewhat unfamiliar • Neither familiar nor unfamiliar • Somewhat familiar • Moderately familiar • Very familiar
15. Have you used ChatGPT? • Yes • No
16. *(If Yes selected)* What was your experience using ChatGPT like?
17. *(Optional)* If there's anything else you'd like to tell us about this study, please enter it here.

A.3 Verbose Descriptions of IPTs Given to Senders

This appendix reports the full text of each IPT as given to senders. Below each IPT are the test cases, with edge cases italicized. We also indicate each IPT's relationship to problems in the humaneval benchmark (Chen et al. [2021]).

1. `max` (*humaneval 35 reworded, including how to specify how to handle an empty list*) You are moving to San Francisco, California. You are looking to rent an apartment and have found a number of similar apartments. You unfortunately have some bad spending habits and already have a lousy credit score of <600, so being able to pay rent as late as possible is a huge plus. Each apartment has a different number of days that you can pay your rent late without penalty. Report the maximum number of days you could potentially pay your rent late. Note that you should make sure to report the maximum, not the minimum or any other number. If there is no maximum number, report -1.
   - Given the task above, and input [30, 40, 10, 50, 25, 40], what should be reported? **[Answer: 50]**
   - Given the task above, and input [100, 200, 300, 400], what should be reported? **[Answer: 400]**
   - *Given the task above, and input [], what should be reported?* **[Answer: -1]**
   - *Given the task above, and input [80, 80, 80, 80], what should be reported?* **[Answer: 80]**

2. `num_even` (*closely related to humaneval 68, 104, 110, and 155; additionally, humaneval 37, 85, 88, 98, 100, 102, 106, 107, 121, 123, 130, 131, 138, and 163 use even-ness as a subroutine*) You are the babysitter of two very troublesome twins who want to eat saltwater taffy. You will be given a list of the number of taffy pieces for a collection of bags of candy. You know that if the pieces in the bag cannot be evenly divided among the twins, the twin that receives fewer pieces will throw a tantrum, which you'd like

to avoid. Report the number of bags that contain an even number of taffy pieces.

- Given the task above, and input [2000, 4003, 10342, 505431], what should be reported? **[Answer: 2]**
- Given the task above, and input [3053, 10643342, 545345, 2879209, 100432, 345082], what should be reported? **[Answer: 3]**
- *Given the task above, and input [-100, -245, 3432, 15432], what should be reported?* **[Answer: 3]**
- *Given the task above, and input [81, 45, 11, 1313, 777, 904329], what should be reported?* **[Answer: 0]**

3. `sum_pos` (*related to humaneval 8, 40, 108, 43, 122, 142, 145, and 151. Less closely related to humaneval 42, 71, 72, 84, 94 114, 121, 128, and 133*) Every summer in July there is an annual Spügelkömpf sports festival. You will be given a list of points athletes scored in a competition. The points are the result of the annual Hugenhömfpfhen competition at the festival. Report the sum of the positive points.

   - Given the task above, and input [3, 5, 7, 10, 2], what should be reported? **[Answer: 27]**
   - Given the task above, and input [100, 5, 20, 543, 12, 1, 6], what should be reported? **[Answer: 687]**
   - *Given the task above, and input [], what should be reported?* **[Answer: 0]**
   - *Given the task above, and input [3.4, -8.2, 1.2, -123, 2.6], what should be reported?* **[Answer: 7.2]**

4. `reignfall` (*no close matches in humaneval*) You are part of a group of friends who like to go out drinking on Friday nights, and who attempt to elect a designated driver democratically. There is a vote between two candidates, "Alice" and "Brooke". Alice has the nicer car, but Brooke always lets you pick the music. A vote for "Alice" is denoted with an "A", while a vote for "Brooke" is denoted with a "B". Votes are held

on Wednesday evenings. The vote was called at a certain cutoff time, denoted "C". You have been chosen by your friends to determine who is the next designated driver. Read the sequence of letters from left to right, stopping the voting count when you see "C". Your friend group uses a sort of buggy app to record votes, so sometimes votes come in after the cutoff time. Count the occurrences of "A"s and "B"s. It is not necessary to count the number of "C"s. If there are the same number of "A"s and "B"s, or more "A"s than "B"s, report "A". You report "A" in this way because Brooke owes you ten bucks, which you are annoyed by. Otherwise, report "B".

- Given the task above, and input ['AABBAAC'], what should be reported? **[Answer: A]**
- Given the task above, and input ['BBAABCBAABCAAC'], what should be reported? **[Answer: B]**
- *Given the task above, and input ['CAABBA'], what should be reported?* **[Answer: A]**
- *Given the task above, and input ['AABBABC'], what should be reported?* **[Answer: A]**

5. `palindrome` (*humaneval 10 reworded*) You are a human in the year 6000. You have recently become friends with a resident of the planet Taerh, which is very similar to Earth. The residents of Taerh are roughly identical to humans, except for the fact that their skin is green, contains chloroplasts, and they photosynthesize energy instead of eating. Your new alien friend is choosing a name to use on both Earth and Taerh, and wants you to verify that the name satisfies a certain property. Your friend wants to choose a new name because they are afraid that humans will make fun of the name they use on Taerh. Report T if the name is a palindrome, F otherwise.

   - Given the task above, and input ['1551'], what should be reported? **[Answer: T]**
   - Given the task above, and input ['MUSIC'], what should be reported? **[Answer:**

F]

- *Given the task above, and input ['-15351'], what should be reported?* **[Answer: F]**
- *Given the task above, and input ['O'], what should be reported?* **[Answer: T]**

6. `find_sum` (*closely related to human-eval 71 and 92*) You have recently decided to drop out of college and pursue your dream of building your own home in the forests of the Pacific Northwest. You are currently attempting to replace a door using scrap wood. You want to use the scrap wood you have lying around because you don't have much in the way of savings. You have a list of scrap wood lengths, and a 'Target' length that the wood must reach to fit in the doorframe. You plan to join the wood together using wood glue, which seeps into the pores of the wood scraps to bind them together. Report a pair of wood lengths from the list, where the sum of the lengths are equal to the 'Target'. We are summing the lengths because joining the wood in this way does not require the wood pieces to overlap for the joint. Report the larger wood length of the pair first. Do not accidentally report the smaller number first. If there is no valid pair, report [].
   - Given the task above, and input [2, 4, 5, 1, 3], 'Target=9', what should be reported? **[Answer: [5, 4]]**
   - Given the task above, and input [1, 6, 9, 3, 5], 'Target=7', what should be reported? **[Answer: [6, 1]]**
   - *Given the task above, and input [4, 4, 3, 1], 'Target=5', what should be reported?* **[Answer: [4, 1]]**
   - *Given the task above, and input [5, 3, 3, 1], 'Target=5', what should be reported?* **[Answer: []]**

7. `rmv_dup` (*humaneval 26 reworded*) Every year, the annual Eidgenössisches Jodlerfest is held in Switzerland, which involves yodeling-related activities. Your job is to manage

a list of contestants for the premier yodeling competition, where every contestant is assigned a number. Contestants are assigned numbers in strictly increasing order. However, some unscrupulous contestants have come back for second and third chances, thinking that you wouldn't recognize them. Unluckily for them, you have eidetic memory. Remove duplicate individuals from the list, but do not otherwise change the list. Removing individuals from the list is equivalent to disqualifying those rule-breakers from participating. Report the list after the duplicates have been removed.

- Given the task above, and input [9, 5, 3, 5], what should be reported? **[Answer: [9, 3]]**
- Given the task above, and input [1, 3, 7, 5, 7], what should be reported? **[Answer: [1, 3, 5]]**
- *Given the task above, and input [3, 3, 3], what should be reported?* **[Answer: []]**
- *Given the task above, and input [25, 25, 1, 10, 10], what should be reported?* **[Answer: [1]]**

8. `str_diff` (*related to humaneval 112*) You have just finished three grueling years of culinary school where you learned to be very organized. You keep recipe cards and abbreviate ingredients to a single letter. You use ISO-size A7 index cards because you think they are the perfect size for an index card. Your assistant has messed up one of your recipe cards by adding an extra ingredient. This guy is always fiddling with your stuff when he thinks you're not looking, which you find very annoying. Report the letter corresponding to the ingredient that has been added to the first recipe card, given the second recipe card that your assistant has modified.

   - Given the task above, and input ['wijht', 'jihqwt'], what should be reported? **[Answer: q]**
   - Given the task above, and input ['wdsffw', 'fpwdsfw'], what should be reported? **[Answer: p]**

- *Given the task above, and input ['ttttttttt', 'tttttttttt'], what should be reported?* **[Answer: t]**
- *Given the task above, and input ['', 'z'], what should be reported?* **[Answer: z]**

9. `is_subseq` (*related to humaneval 7, 18, 29, and 154*) You are a poet making poetry out of old books. You just came across a first edition copy of "Manfred" by George Gordon Byron, 6th Baron Byron, also known as Lord Byron, one of the finest English poets to ever live (in your opinion), and an inspiration to your poetry. You are given two sequences of letters: one is the letters of a poem that you want to write. Specifically, after coming across Manfred's monologue at the beginning of Scene II, you decide that the poem you want to write is a permutation of his inspiring speech. The second sequence is letters on the page of the book. Letters correspond to phonemes, a fundamental linguistic building block. Given the two sequences, determine if you can form your poem by (optionally) deleting letters from the pages of the book. You remove letters by physically cutting them out of the page with your handy penknife. Report 'T' if you can make your poem, and report 'F' if you cannot.

   - Given the task above, and input ['abc', 'ahbgdc'], what should be reported? **[Answer: T]**
   - Given the task above, and input ['bequtie', 'buth'], what should be reported? **[Answer: F]**
   - *Given the task above, and input ['cbd', 'cxwdpbu'], what should be reported?* **[Answer: F]**
   - *Given the task above, and input ['uit', 'uttiuiuut'], what should be reported?* **[Answer: T]**

10. `exact_chg` (*no close match in humaneval*) It is Friday, November 22, 1963, in Dallas, Texas. You are trying to take a bus ride, but need to have exact change to board. Little do you know that today is the day that John F. Kennedy, the 35th President of

the United States, will be assassinated. You have three numbers, respectively named ‘Small,’ ‘Big,’ and ‘Goal.’ Each of these numbers represent something. ‘Small’ represents the number of one dollar bills you have. You think one dollar bills are ok, and that they are overall much more useful than pennies. ‘Big’ represents the number of five dollar bills you have. You actually quite like five dollar bills because you enjoy counting by fives. ‘Goal’ represents the cost of the bus ride. The buses in Dallas are quite expensive for reasons no one understands. Determine if you can take the bus with the change you have. You need exact change because otherwise the bus driver will not let you board. Report ‘T’ if you have exact change for the amount of money equal to ‘Goal,’ and ‘F’ otherwise.

- Given the task above, and input ['Small=3', 'Big=1', 'Goal=9'], what should be reported? **[Answer: F]**
- Given the task above, and input ['Small=6', 'Big=1', 'Goal=4'], what should be reported? **[Answer: T]**
- *Given the task above, and input ['Small=2', 'Big=2', 'Goal=8'], what should be reported?* **[Answer: F]**
- *Given the task above, and input ['Small=5', 'Big=4', 'Goal=10'], what should be reported?* **[Answer: T]**

11. `fizzbuzz` (*humaneval 36 reworded*) You are a spy in 1967, during the height of the Cold War, and have been sent to Leningrad undercover. As a spy, you are attempting to defuse a bomb. The bomb is a BLU-3 Pineapple Cluster Bomblet, which you have studied the properties of extensively. The bomb displays a number on its main panel, "n". You're currently on a mission to infiltrate a suspected double agent's office, but the Soviets must be onto you as they are currently swarming the building–you'll need to defuse the bomb, and get out ASAP! Given the value of n, you must report the number that defuses the bomb. Otherwise your life will be brought to an unfortunate

end. Count and report the number of times the digit 7 appears in whole numbers less than n that are divisible by 11 OR 13. Troublingly, you find checking divisibility by 11 and 13 to be noticeably more difficult than checking divisibility by 2 or 5. Do not "double-count" if the number is divisible by both 11 AND 13.

- Given the task above, and input ['n=130'], what should be reported? **[Answer: 4]**
- Given the task above, and input ['n=8'], what should be reported? **[Answer: 0]**
- *Given the task above, and input ['n=79.777'], what should be reported?* **[Answer: 3]**
- *Given the task above, and input ['n=80.0'], what should be reported?* **[Answer: 3]**

12. `xyz` (*humaneval 159 reworded*) You're a hungry rabbit. You are also highly intelligent for a rabbit, and have the ability to make precise plans for the future. You want to 'eat(x, y, z)' based on three factors: 'x=the number of carrots you have already eaten', 'y=the number of carrots you want to eat', and 'z=the number of carrots in your fridge'. You have also reinvented the refrigerator and set your fridge at the optimal temperature for storing carrots. Report a pair of numbers: [total number of carrots eaten, carrots left in the fridge]. These are genetically engineered carrots that don't spoil. The total number of carrots left in the fridge is equal to the number of carrots in the fridge minus the number of carrots you want to eat, or 0–whichever is greater. Carrots are actually not the healthiest food for rabbits. The total number of carrots that you've eaten is equal to the number of carrots you've already eaten, plus any carrots you eat today.

    - Given the task above, and input ['(10, 3, 20)'], what should be reported? **[Answer: [13, 17]]**
    - Given the task above, and input ['(2, 5, 5)'], what should be reported? **[Answer:**

[7, 0]]

- *Given the task above, and input ['(0, 4, 0)'], what should be reported?* **[Answer: [0, 0]]**
- *Given the task above, and input ['(12, 0, 50)'], what should be reported?* **[Answer: [12, 50]]**

A.4 Codebook Developed For Qualitative Analysis

Qualitative Code	Description	Example
All information	Contains all the necessary information to solve the test cases	From left to right, count the number of As and Bs and stopping when you see a C. Report whichever letter has the greater number. If there is tie, report A.
Some information	Missing some of the necessary information to solve the test cases	Count the number of As and Bs and report whichever letter has the greater number.
No Information	Contains none of the necessary information to solve testcases (may simply repeat the context)	We are having a vote between my friends
Generalized All	The procedure is fully generalizable; none of the context remains	Count if there are more As or Bs
Generalized Some	The procedure has been somewhat generalized, but some context remains (e.g., given participant's score totals, find the max)	Count whether A or B has more votes to be driver.
Generalized None	The procedure has not been generalized in any significant way, it is highly tied to its context (e.g., there is an annual compitition...participants score based on x y z... given a list of score totals, report the maximum score)	You are holding a vote between your friends Alice and Brooke. Count whether Alice or Brooke has more votes to be designated driver.
Correct Examples	The procedure contains examples not corresponding to edge-cases; the examples are correct	Example: "ABAABAC" Answer: "A"
Incorrect Examples	The procedure contains examples not corresponding to edge-cases; but one or more examples is incorrect	Example: "ABAABAC" Answer: "B"
Edge Example Correct	The procedure contains one or more examples corresponding to an edge-case; the edge-case examples are correct	Example: "CABAABBBBB" Answer: "A"
Edge Example Incorrect	The procedure contains one or more examples corresponding to an edge-case; but one or more examples is incorrect	Example: "CABAABBBBB" Answer: "B"
Contains Extra Information	The procedure contains extraneous information	Count whether Alice or Brooke has more votes to be designated driver. Alice has the nicer car but Brooke lets you pick the music.
Concise Attempt	A (noticeable) attempt was made to make the procedure more concise	Read from left to right and report the letter that appears more before the first C, if there is a tie, report A.
Steps	The procedure includes steps to solve the task	Read the sequence from left to right. Then keep tally of how many As and Bs there are. Stop tallying when you see a C. Then compare the frequencies of A and B. Report the letter that corresponds to the higher frequency.
No Meaningful Information	The procedure is devoid of any meaningful information	You're holding a vote between your friends to pick a driver.... shouldn't driving be a chore?
ChatGPT	The procedure seems to have been written by ChatGPT or another AI tool	Certainly! Here is a simplified algorithm procedure:
Exact Match	The participant more or less exactly re-wrote the procedure the researchers (us) provided; if the entire structure is the same, and minor to no words changed, we mark it as Exact Match because there is no evidence that the sender processed the procedure. If the participant added examples, we consider the procedure changed enough that it is NOT an Exact Match	I'm in of a group of friends who like to go out drinking on Friday nights I am attempt to elect a designated driver democratically. There is a vote between two candidates, "Alice" and "Brooke". Alice has the nicer car, but Brooke always lets me pick the music. A vote for "Alice" is denoted with an 'A', while a vote for "Brooke" is denoted with a 'B'. Votes are held on Wednesday evenings. The vote was called at a certain cutoff time, denoted 'C'. I have been chosen by my friends to determine who is the next designated driver. Read the sequence of letters from left to right, stopping the voting count when you see 'C'. We use a sort of buggy app to record votes, so sometimes votes come in after the cutoff time. Count the occurrences of 'A's and 'B's. It is not necessary to count the number of 'C's. If there are the same number of 'A's and 'B's, or more 'A's than 'B's, report 'A'. You report 'A' in this way because Brooke owes me ten bucks, which I'm annoyed by. Otherwise, report 'B'.

A.5 Additional Graphs of Demographic Correlations

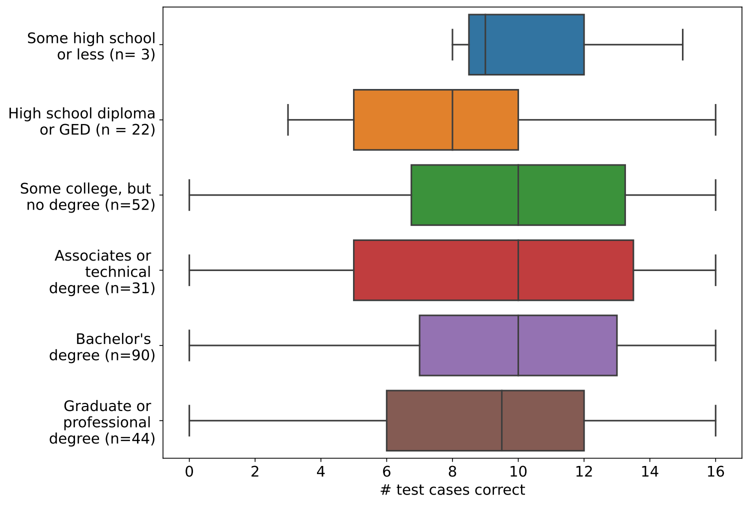


Figure A.1: The number of test cases answered correctly split by participants' highest level of education.

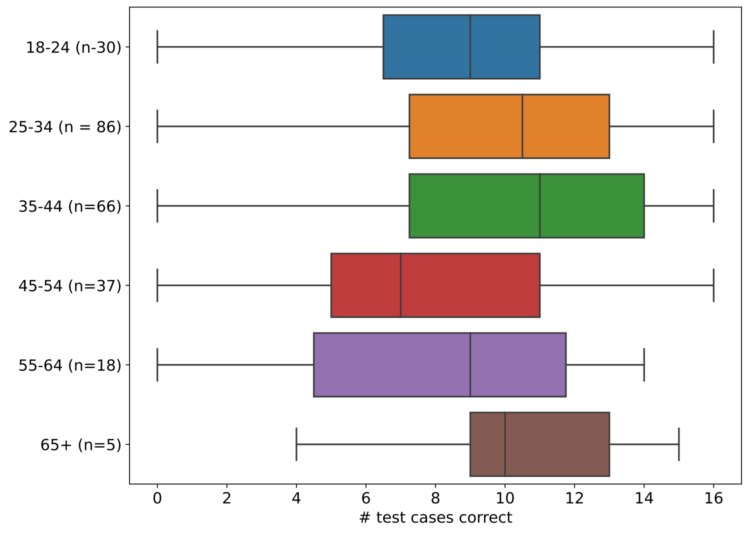


Figure A.2: The number of test cases answered correctly split by participants' age range.

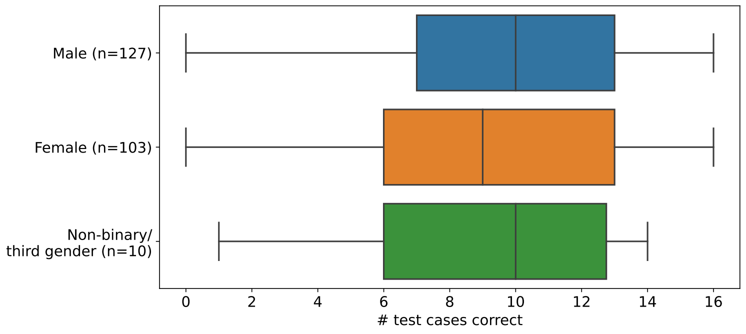


Figure A.3: The number of test cases answered correctly split by participants' gender.

A.6 Additional Regression Tables for LLMs

Table A.2: Linear regression with the dependent variable indicating how many of the four test cases *Code Llama 34B* answered correctly when *directly answering* test cases. $R^2 = 0.323$.

Independent Variable	**Baseline**	β	**SE**	**t**	**p**
(Intercept)	–	2.158	0.215	10.040	$<$**.001**
Sender is Programmer	Non-programmer	0.281	0.127	2.209	**.028**
Sender Age 18–24	25–34	-0.179	0.187	-0.955	.340
Sender Age 35–44	25–34	-0.070	0.130	-0.535	.593
Sender Age 44+	25–34	-0.557	0.151	-3.683	$<$**.001**
Sender is Non-male	Male	0.021	0.114	0.187	.852
Sender Degree: None	Bachelor's	0.082	0.128	0.640	.522
Sender Degree: Associate's	Bachelor's	0.210	0.192	1.094	.274
Sender Degree: Graduate	Bachelor's	0.254	0.157	1.614	.107
Condition: Examples	Non-examples	-0.066	0.140	-0.474	.635
Examples Sent	None Sent	0.475	0.170	2.799	**.005**
Condition: Interactive	Non-interactive	0.010	0.117	0.089	.929
Participants Interacted	Did Not	-0.370	0.212	-1.743	.082
Interaction Coded As Useful	Was Not	0.757	0.282	2.686	**.007**
IPT: `num_even`	`max`	-1.391	0.251	-5.542	$<$**.001**
IPT: `sum_pos`	`max`	0.050	0.245	0.203	.839
IPT: `reignfall`	`max`	0.359	0.266	1.350	.178
IPT: `palindrome`	`max`	0.076	0.242	0.316	.752
IPT: `find_sum`	`max`	-0.957	0.251	-3.811	$<$**.001**
IPT: `rmv_dup`	`max`	-1.664	0.239	-6.966	$<$**.001**
IPT: `str_diff`	`max`	-0.895	0.241	-3.709	$<$**.001**
IPT: `is_subseq`	`max`	-0.040	0.246	-0.162	.872
IPT: `exact_chg`	`max`	-0.222	0.237	-0.938	.349
IPT: `fizzbuzz`	`max`	-1.666	0.243	-6.866	$<$**.001**
IPT: `xyz`	`max`	-1.094	0.252	-4.346	$<$**.001**

Table A.3: Linear regression with the dependent variable indicating how many of the four test cases **Code Llama 34B** answered correctly when *producing Python code*. $R^2 = 0.457$.

Independent Variable	**Baseline**	β	**SE**	**t**	**p**
(Intercept)	–	1.480	0.235	6.302	<**.001**
Sender is Programmer	Non-programmer	0.499	0.139	3.589	<**.001**
Sender Age 18–24	25–34	-0.036	0.205	-0.176	.860
Sender Age 35–44	25–34	-0.118	0.142	-0.832	.406
Sender Age 44+	25–34	-0.632	0.165	-3.822	<**.001**
Sender is Non-male	Male	-0.012	0.124	-0.094	.925
Sender Degree: None	Bachelor's	0.112	0.139	0.805	.421
Sender Degree: Associate's	Bachelor's	0.081	0.210	0.386	.700
Sender Degree: Graduate	Bachelor's	0.113	0.172	0.659	.510
Condition: Examples	Non-examples	0.247	0.153	1.611	.108
Examples Sent	None Sent	0.518	0.185	2.795	**.005**
Condition: Interactive	Non-interactive	0.079	0.128	0.620	.536
Participants Interacted	Did Not	-0.416	0.232	-1.793	.074
Interaction Coded As Useful	Was Not	-0.005	0.308	-0.015	.988
IPT: `num_even`	`max`	0.362	0.274	1.321	.187
IPT: `sum_pos`	`max`	1.670	0.268	6.239	<**.001**
IPT: `reignfall`	`max`	0.356	0.291	1.223	.222
IPT: `palindrome`	`max`	1.843	0.264	6.983	<**.001**
IPT: `find_sum`	`max`	-0.711	0.274	-2.593	**.010**
IPT: `rmv_dup`	`max`	-1.745	0.261	-6.687	<**.001**
IPT: `str_diff`	`max`	-1.041	0.264	-3.950	<**.001**
IPT: `is_subseq`	`max`	-0.326	0.268	-1.214	.226
IPT: `exact_chg`	`max`	-0.128	0.259	-0.494	.621
IPT: `fizzbuzz`	`max`	-0.938	0.265	-3.537	<**.001**
IPT: `xyz`	`max`	-0.212	0.275	-0.772	.441

Table A.4: Linear regression with the dependent variable indicating how many of the four test cases *Gemini* answered correctly when *directly answering* test cases. $R^2 = 0.429$.

Independent Variable	**Baseline**	β	**SE**	**t**	**p**
(Intercept)	–	2.391	0.194	12.308	<**.001**
Sender is Programmer	Non-programmer	0.071	0.115	0.619	.536
Sender Age 18–24	25–34	0.058	0.169	0.344	.731
Sender Age 35–44	25–34	-0.102	0.118	-0.865	.387
Sender Age 44+	25–34	-0.294	0.137	-2.151	**.032**
Sender is Non-male	Male	-0.133	0.103	-1.293	.197
Sender Degree: None	Bachelor's	0.078	0.115	0.677	.499
Sender Degree: Associate's	Bachelor's	-0.209	0.173	-1.207	.228
Sender Degree: Graduate	Bachelor's	0.030	0.142	0.214	.831
Condition: Examples	Non-examples	0.054	0.127	0.428	.669
Examples Sent	None Sent	0.653	0.153	4.260	<**.001**
Condition: Interactive	Non-interactive	0.045	0.106	0.422	.673
Participants Interacted	Did Not	0.080	0.192	0.418	.676
Interaction Coded As Useful	Was Not	0.089	0.255	0.350	.727
IPT: `num_even`	`max`	-1.944	0.227	-8.570	<**.001**
IPT: `sum_pos`	`max`	0.283	0.222	1.279	.202
IPT: `reignfall`	`max`	-1.901	0.241	-7.898	<**.001**
IPT: `palindrome`	`max`	-2.035	0.218	-9.316	<**.001**
IPT: `find_sum`	`max`	-0.621	0.227	-2.736	.006
IPT: `rmv_dup`	`max`	-1.826	0.216	-8.454	<**.001**
IPT: `str_diff`	`max`	-1.752	0.218	-8.036	<**.001**
IPT: `is_subseq`	`max`	-1.503	0.222	-6.765	<**.001**
IPT: `exact_chg`	`max`	-2.152	0.214	-10.039	<**.001**
IPT: `fizzbuzz`	`max`	-1.945	0.219	-8.867	<**.001**
IPT: `xyz`	`max`	-1.430	0.228	-6.287	<**.001**

Table A.5: Linear regression with the dependent variable indicating how many of the four test cases *Gemini* answered correctly when *producing Python code*. $R^2 = 0.406$.

Independent Variable	**Baseline**	β	**SE**	**t**	**p**
(Intercept)	–	1.483	0.253	5.853	<**.001**
Sender is Programmer	Non-programmer	0.430	0.150	2.866	**.004**
Sender Age 18–24	25–34	-0.213	0.221	-0.964	.335
Sender Age 35–44	25–34	-0.143	0.153	-0.932	.352
Sender Age 44+	25–34	-0.825	0.178	-4.626	<**.001**
Sender is Non-male	Male	-0.013	0.134	-0.094	.925
Sender Degree: None	Bachelor's	0.038	0.150	0.252	.801
Sender Degree: Associate's	Bachelor's	0.021	0.226	0.094	.925
Sender Degree: Graduate	Bachelor's	0.201	0.185	1.082	.280
Condition: Examples	Non-examples	0.062	0.165	0.374	.709
Examples Sent	None Sent	0.753	0.200	3.768	<**.001**
Condition: Interactive	Non-interactive	-0.133	0.138	-0.963	.336
Participants Interacted	Did Not	-0.301	0.250	-1.202	.230
Interaction Coded As Useful	Was Not	-0.033	0.332	-0.100	.921
IPT: `num_even`	`max`	1.153	0.296	3.897	<**.001**
IPT: `sum_pos`	`max`	1.778	0.289	6.156	<**.001**
IPT: `reignfall`	`max`	0.769	0.314	2.451	**.015**
IPT: `palindrome`	`max`	2.048	0.285	7.185	<**.001**
IPT: `find_sum`	`max`	0.099	0.296	0.336	.737
IPT: `rmv_dup`	`max`	-1.444	0.282	-5.126	<**.001**
IPT: `str_diff`	`max`	-0.690	0.284	-2.424	**.016**
IPT: `is_subseq`	`max`	0.427	0.290	1.473	.141
IPT: `exact_chg`	`max`	0.115	0.280	0.412	.680
IPT: `fizzbuzz`	`max`	-0.495	0.286	-1.730	.084
IPT: `xyz`	`max`	0.0360	0.297	0.121	.904

REFERENCES

ACM Visual Identity Standards. Association for Computing Machinery, 2007. `http://identitystandards.acm.org`.

Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, et al. Program synthesis with large language models, 2021.

Rebecca Balebako, Abigail Marsh, Jialiu Lin, Jason Hong, and Lorrie Cranor. The privacy and security behaviors of smartphone app developers, 01 2014.

Barbara Rita Barricelli, Fabio Cassano, Daniela Fogli, and Antonio Piccinno. End-user development, end-user programming and end-user software engineering: A systematic mapping study. *Journal of Systems and Software*, 149:101–137, 2019.

Brett A Becker, Paul Denny, James Finnie-Ansley, Andrew Luxton-Reilly, James Prather, and Eddie Antonio Santos. Programming is hard–or at least it used to be: Educational opportunities and challenges of ai code generation. *arXiv preprint arXiv:2212.01020*, 2022.

BigScience Workshop. Bloom: A 176b-parameter open-access multilingual language model, 2023.

Johannes L. Braams and Javier Bezos. *Babel*, 2022. `http://www.ctan.org/pkg/babel`.

Rogério Brito. *The algorithms bundle*, August 2009. `http://www.ctan.org/pkg/algorithms`.

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners, 2020. URL `https://arxiv.org/abs/2005.14165`.

Yuzhe Cai, Shaoguang Mao, Wenshan Wu, Zehua Wang, Yaobo Liang, Tao Ge, Chenfei Wu, Wang You, Ting Song, Yan Xia, et al. Low-code llm: Visual programming over llms. *arXiv:2304.08103*, 2023.

Sarina Canelake. A gentle introduction to programming using python, 2011. URL `https://ocw.mit.edu/courses/6-189-a-gentle-introduction-to-programming-using-python-january-iap-2011/`.

David Carlisle. *The textcase package*, October 2004. `http://www.ctan.org/pkg/textcase`.

Zijian Ding Chan et al. Mapping the design space of interactions in human-ai text co-creation tasks. *arXiv preprint arXiv:2303.06430*, 2023.

Mark Chen et al. Evaluating large language models trained on code, 2021. URL `https://arxiv.org/abs/2107.03374`.

Ruijia Cheng, Alison Smith-Renner, Ke Zhang, Joel Tetreault, and Alejandro Jaimes-Larrarte. Mapping the design space of human-AI interaction in text summarization. In *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 431–455, Seattle, United States, July 2022. Association for Computational Linguistics. doi:10.18653/v1/2022.naacl-main.33. URL `https://aclanthology.org/2022.naacl-main.33`.

Aakanksha Chowdhery et al. Palm: Scaling language modeling with pathways, 2022.

Matteo Ciniselli, Nathan Cooper, Luca Pascarella, Antonio Mastropaolo, Emad Aghajani, Denys Poshyvanyk, Massimiliano Di Penta, and Gabriele Bavota. An empirical study on the usage of transformer models for code completion, 2021. URL `https://doi.org/10.1109%2Ftse.2021.3128234`.

Arghavan Moradi Dakhel, Vahid Majdinasab, Amin Nikanjam, Foutse Khomh, Michel C. Desmarais, Zhen Ming, and Jiang. Github copilot ai pair programmer: Asset or liability?, 2023.

Anastasia Danilova, Alena Naiakshina, Stefan Horstmann, and Matthew Smith. Do you really code? designing and evaluating screening questions for online surveys with programmers, 2021.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding, 2019.

Michael Downes and Barbara Beeton. *The amsart, amsproc, and amsbook document classes*. American Mathematical Society, August 2004. `http://www.ctan.org/pkg/amslatex`.

Jean-Baptiste Döderlein, Mathieu Acher, Djamel Eddine Khelladi, and Benoit Combemale. Piloting copilot and codex: Hot temperature, cold prompts, or black magic?, 2023.

Hugo Touvron et al. Llama 2: Open foundation and fine-tuned chat models, 2023.

Simon Fear. *Publication quality tables in LaTeX*, April 2005. `http://www.ctan.org/pkg/booktabs`.

James Finnie-Ansley, Paul Denny, Brett A Becker, Andrew Luxton-Reilly, and James Prather. The robots are coming: Exploring the implications of openai codex on introductory programming, 2022.

James Finnie-Ansley, Paul Denny, Andrew Luxton-Reilly, Eddie Antonio Santos, James Prather, and Brett A Becker. My ai wants to know if this will be on the exam: Testing openai's codex on cs2 programming exercises, 2023.

Cristophe Fiorio. *algorithm2e.sty—package for algorithms*, October 2015. `http://www.ctan.org/pkg/algorithm2e`.

Neil Fraser. Ten things we've learned from blockly. In *2015 IEEE blocks and beyond workshop (blocks and beyond)*, pages 49–50, 2015.

Simret Araya Gebreegziabher, Zheng Zhang, Xiaohang Tang, Yihao Meng, Elena Glassman, and Toby Jia-Jun Li. Patat: Human-ai collaborative qalitative coding with explainable interactive rule synthesis. *corpora*, 7(27):52, 2023.

GitHub. Copilot: Your ai pair programmer, 2022. URL `https://github.com/features/copilot`.

Gemini Team Google. Gemini: A family of highly capable multimodal models, 2023.

Rahul Gupta, Soham Pal, Aditya Kanade, and Shirish Shevade. Deepfix: Fixing common c language errors by deep learning, Feb. 2017. URL `https://ojs.aaai.org/index.php/AAAI/article/view/10742`.

Carsten Heinz, Brooks Moses, and Jobst Hoffmann. *The Listings Package*, June 2015. `http://www.ctan.org/pkg/listings`.

Dan Hendrycks, Steven Basart, Saurav Kadavath, Mantas Mazeika, Akul Arora, Ethan Guo, Collin Burns, Samir Puranik, Horace He, Dawn Song, and Jacob Steinhardt. Measuring coding challenge competence with apps, 2021.

Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de Las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, Tom Hennigan, Eric Noland, Katie Millican, George van den Driessche, Bogdan Damoc, Aurelia Guy, Simon Osindero, Karen Simonyan, Erich Elsen, Jack W. Rae, Oriol Vinyals, and Laurent Sifre. Training compute-optimal large language models, 2022.

Justin Huang and Maya Cakmak. Supporting mental model accuracy in trigger-action programming. In *Proceedings of the 2015 acm international joint conference on pervasive and ubiquitous computing*, pages 215–225, 2015.

Dhanya Jayagopal, Justin Lubin, and Sarah E. Chasins. Exploring the learnability of program synthesizers by novice programmers. In *Proceedings of the 35th Annual ACM Symposium on User Interface Software and Technology*, UIST '22, New York, NY, USA, 2022a. Association for Computing Machinery. ISBN 9781450393201. doi:10.1145/3526113.3545659. URL `https://doi.org/10.1145/3526113.3545659`.

Dhanya Jayagopal, Justin Lubin, and Sarah E. Chasins. Exploring the learnability of program synthesizers by novice programmers. In *Proceedings of the 35th Annual ACM Symposium on User Interface Software and Technology*, UIST '22, New York, NY, USA, 2022b. Association for Computing Machinery. ISBN 9781450393201. doi:10.1145/3526113.3545659. URL `https://doi.org/10.1145/3526113.3545659`.

Eirini Kalliamvakou. Research: quantifying github copilot's impact on developer productivity and happiness, 2022. URL `https://github.blog/2022-09-07-research-quantifying-github-copilots-impact-on-developer-productivity-and-happiness/`.

Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B. Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. Scaling laws for neural language models, 2020.

Ulas Berk Karli, Juo-Tung Chen, Victor Nikhil Antony, and Chien-Ming Huang. Alchemist: Llm-aided end-user development of robot applications. In *Proceedings of the 2024 ACM/IEEE International Conference on Human-Robot Interaction*, pages 361–370, 2024.

Mohammad Amin Kuhail, Shahbano Farooq, Rawad Hammad, and Mohammed Bahja. Characterizing visual programming approaches for end-user developers: A systematic review. *IEEE Access*, 9:14181–14202, 2021.

Mina Lee, Megha Srivastava, Amelia Hardy, John Thickstun, Esin Durmus, Ashwin Paranjape, Ines Gerard-Ursin, Xiang Lisa Li, Faisal Ladhak, Frieda Rong, Rose E. Wang, Minae Kwon, Joon Sung Park, Hancheng Cao, Tony Lee, Rishi Bommasani, Michael Bernstein, and Percy Liang. Evaluating human-language model interaction, 2022. URL `https://arxiv.org/abs/2212.09746`.

Mina Lee, Megha Srivastava, Amelia Hardy, John Thickstun, Esin Durmus, Ashwin Paranjape, Ines Gerard-Ursin, Xiang Lisa Li, Faisal Ladhak, Frieda Rong, Rose E. Wang, Minae Kwon, Joon Sung Park, Hancheng Cao, Tony Lee, Rishi Bommasani, Michael Bernstein, and Percy Liang. Evaluating human-language model interaction, 2024.

LeetCode. Leetcode: A new way to learn, 2023. URL `https://leetcode.com/`.

Nicola Leonardi, Marco Manca, Fabio Paternò, and Carmen Santoro. Trigger-action programming for personalising humanoid robot behaviour. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*, pages 1–13, 2019.

Yujia Li, David Choi, Junyoung Chung, Nate Kushman, Julian Schrittwieser, Rémi Leblond, Tom Eccles, James Keeling, Felix Gimeno, Agustin Dal Lago, et al. Competition-level code generation with alphacode. *Science*, 378(6624):1092–1097, 2022.

Henry Lieberman, Fabio Paternò, Markus Klann, and Volker Wulf. End-user development: An emerging paradigm. In *End user development*, pages 1–8. Springer, 2006.

Michael Xieyang Liu, Advait Sarkar, Carina Negreanu, Benjamin Zorn, Jack Williams, Neil Toronto, and Andrew D. Gordon. "what it wants me to say": Bridging the abstraction gap between end-user programmers and code-generating large language models, apr 2023. URL `https://doi.org/10.1145%2F3544548.3580817`.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. Roberta: A robustly optimized bert pretraining approach, 2019.

Andrew M McNutt, Chenglong Wang, Robert A Deline, and Steven M Drucker. On the design of ai-powered code assistants for notebooks, 2023.

André N. Meyer, Earl T. Barr, Christian Bird, and Thomas Zimmermann. Today was a good day: The daily life of software developers. *IEEE Transactions on Software Engineering*, 47(5):863–880, 2021. doi:10.1109/TSE.2019.2904957.

Xianghang Mi, Feng Qian, Ying Zhang, and XiaoFeng Wang. An empirical characterization of ifttt: ecosystem, usage, and performance. In *Proceedings of the 2017 Internet Measurement Conference*, pages 398–404, 2017.

Johan Moreno. Openai positioned itself as the ai leader in 2022. but could google supersede it in '23?, Dec 2022. URL `https://www.forbes.com/sites/johanmoreno/2022/12/29/openai-positioned-itself-as-the-ai--leader-in-2022-but-could-google-supersede-it-in-23/?sh=75f491153216`.

Brad A Myers, Amy J Ko, and Margaret M Burnett. Invited research overview: end-user programming. In *CHI'06 extended abstracts on Human factors in computing systems*, pages 75–80, 2006.

Bonnie A Nardi. *A small matter of programming: perspectives on end user computing*. MIT press, 1993.

Nhan Nguyen and Sarah Nadi. An empirical evaluation of github copilot's code suggestions. In *Proceedings of the 19th International Conference on Mining Software Repositories*, pages 1–5, 2022.

Erik Nijkamp, Bo Pang, Hiroaki Hayashi, Lifu Tu, Huan Wang, Yingbo Zhou, Silvio Savarese, and Caiming Xiong. Codegen: An open large language model for code with multi-turn program synthesis. *arXiv preprint arXiv:2203.13474*, 2022.

OpenAI. Introducing chatgpt, 2022. URL `https://openai.com/blog/chatgpt`.

OpenAI. Gpt-4 technical report, 2024.

Hammond Pearce, Baleegh Ahmad, Benjamin Tan, Brendan Dolan-Gavitt, and Ramesh Karri. Asleep at the keyboard? assessing the security of github copilot's code contributions, 2021.

Neil Perry, Megha Srivastava, Deepak Kumar, and Dan Boneh. Do users write more insecure code with ai assistants?, 2022. URL `https://arxiv.org/abs/2211.03622`.

Prolific. Prolific: A higher standard of online research, 2023. URL `https://www.prolific.co/`.

Alec Radford, Jeff Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners, 2019.

Amir Rahmati, Earlence Fernandes, Jaeyeon Jung, and Atul Prakash. Ifttt vs. zapier: A comparative study of trigger-action programming frameworks. *arXiv:1709.02788*, 2017.

Mitchel Resnick, John Maloney, Andrés Monroy-Hernández, Natalie Rusk, Evelyn Eastmond, Karen Brennan, Amon Millner, Eric Rosenbaum, Jay Silver, Brian Silverman, et al. Scratch: programming for all. *Communications of the ACM*, 52(11):60–67, 2009.

Kevin Roose. How schools can survive (and maybe even thrive) with a.i. this fall, 2024. URL `https://www.nytimes.com/2023/08/24/technology/how-schools-can-survive-and-maybe-even-thrive-with-ai-this-fall.html`.

Steven I. Ross, Fernando Martinez, Stephanie Houde, Michael Muller, and Justin D. Weisz. The programmer's assistant: Conversational interaction with a large language model for software development, mar 2023. URL `https://doi.org/10.1145%2F3581641.3584037`.

Baptiste Rozière, Jonas Gehring, Fabian Gloeckle, Sten Sootla, Itai Gat, Xiaoqing Ellen Tan, Yossi Adi, Jingyu Liu, Romain Sauvestre, Tal Remez, Jérémy Rapin, Artyom Kozhevnikov, Ivan Evtimov, Joanna Bitton, Manish Bhatt, Cristian Canton Ferrer, Aaron Grattafiori, Wenhan Xiong, Alexandre Défossez, Jade Copet, Faisal Azhar, Hugo Touvron, Louis Martin, Nicolas Usunier, Thomas Scialom, and Gabriel Synnaeve. Code llama: Open foundation models for code, 2024.

Gustavo Sandoval, Hammond Pearce, Teo Nys, Ramesh Karri, Siddharth Garg, and Brendan Dolan-Gavitt. Lost at c: A user study on the security implications of large language model code assistants. In *32nd USENIX Security Symposium (USENIX Security 23)*, pages 2205–2222, Anaheim, CA, August 2023. USENIX Association. ISBN 978-1-939133-37-3. URL `https://www.usenix.org/conference/usenixsecurity23/presentation/sandoval`.

Sami Sarsa, Paul Denny, Arto Hellas, and Juho Leinonen. Automatic generation of programming exercises and code explanations using large language models. In *Proceedings of the 2022 ACM Conference on International Computing Education Research - Volume 1*, ICER '22, page 27–43, New York, NY, USA, 2022. Association for Computing Machinery. ISBN 9781450391948. doi:10.1145/3501385.3543957. URL `https://doi.org/10.1145/3501385.3543957`.

Axel Sommerfeldt. *The subcaption package*, April 2013. `http://www.ctan.org/pkg/subcaption`.

Aarohi Srivastava et al. Beyond the imitation game: Quantifying and extrapolating the capabilities of language models, 2022. URL `https://arxiv.org/abs/2206.04615`.

Nicola L. C. Talbot. *User Manual for glossaries.sty v4.44*, December 2019. `http://www.ctan.org/pkg/glossaries`.

Florian Tambon, Arghavan Moradi Dakhel, Amin Nikanjam, Foutse Khomh, Michel C. Desmarais, and Giuliano Antoniol. Bugs in large language models generated code: An empirical study, 2024.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurelien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. Llama: Open and efficient foundation language models, 2023.

Immanuel Trummer. Codexdb: Generating code for processing sql queries using gpt-3 codex, 2022. URL `https://arxiv.org/abs/2204.08941`.

UK TEX Users Group. UK list of TEX frequently asked questions. `https://texfaq.org`, 2019.

Blase Ur, Elyse McManus, Melwyn Pak Yong Ho, and Michael L Littman. Practical trigger-action programming in the smart home. In *Proceedings of the SIGCHI conference on human factors in computing systems*, pages 803–812, 2014.

Blase Ur, Melwyn Pak Yong Ho, Stephen Brawner, Jiyun Lee, Sarah Mennicken, Noah Picard, Diane Schulze, and Michael L Littman. Trigger-action programming in the wild: An analysis of 200,000 ifttt recipes. In *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems*, pages 3227–3231, 2016.

Priyan Vaithilingam, Tianyi Zhang, and Elena L. Glassman. Expectation vs. experience: Evaluating the usability of code generation tools powered by large language models, 2022. URL `https://doi.org/10.1145/3491101.3519665`.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2017. URL `https://arxiv.org/abs/1706.03762`.

Boris Veytsman, Bern Schandl, Lee Netherton, and C. V. Radhakrishnan. *A package to create a nomenclature*, September 2005. `http://www.ctan.org/pkg/nomencl`.

Jeffrey Wong and Jason I Hong. Making mashups with marmite: towards end-user programming for the web. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 1435–1444, 2007.

Tongshuang Wu, Michael Terry, and Carrie J. Cai. Ai chains: Transparent and controllable human-ai interaction by chaining large language model prompts, 2021. URL `https://arxiv.org/abs/2110.01691`.

Qualtrics XM. Qualtrics: Better frontlines. better bottom line., 2023. URL `https://www.qualtrics.com/`.