How Humans Communicate Programming Tasks in Natural Language and Implications For End-User Programming with LLMs

Madison Pickering
University of Chicago
Chicago, Illinois, USA
madisonp@uchicago.edu

Helena Williams
University of Chicago
Chicago, Illinois, USA
hwilliams10@uchicago.edu

Alison Gan
University of Chicago
Chicago, Illinois, USA
agan@uchicago.edu

Weijia He
University of Southampton
Southampton, United Kingdom
University of Chicago
Chicago, Illinois, USA
weijia.he@soton.ac.uk

Hyojae Park
Carnegie Mellon University
Pittsburgh, Pennsylvania, USA
hyojae@cmu.edu

Francisco Piedrahita Velez
Brown University
Providence, Rhode Island, USA
fpiedrah@brown.edu

Michael L. Littman
Brown University
Providence, Rhode Island, USA
mlittman@cs.brown.edu

Blase Ur
University of Chicago
Chicago, Illinois, USA
blase@uchicago.edu

Abstract

Large language models (LLMs) like GPT-4 can convert natural-language descriptions of a task into computer code, making them a promising interface for end-user programming. We undertake a systematic analysis of how people with and without programming experience describe information-processing tasks (IPTs) in natural language, focusing on the characteristics of successful communication. Across two online between-subjects studies, we paired crowdworkers either with one another or with an LLM, asking senders (always humans) to communicate IPTs in natural language to their receiver (either a human or LLM). Both senders and receivers tried to answer test cases, the latter based on their sender's description. While participants with programming experience tended to communicate IPTs more successfully than non-programmers, this advantage was not overwhelming. Furthermore, a user interface that solicited example test cases from senders often, but not always, improved IPT communication. Allowing receivers to request clarification, though, was less successful at improving communication.

CCS Concepts

• **Human-centered computing → Empirical studies in interaction design**.

Keywords

Large Language Models, LLMs, End-User Programming

CHI '25, Yokohama, Japan

ACM ISBN 979-8-4007-1394-1/25/04
https://doi.org/10.1145/3706598.3713271

ACM Reference Format:
Madison Pickering, Helena Williams, Alison Gan, Weijia He, Hyojae Park, Francisco Piedrahita Velez, Michael L. Littman, and Blase Ur. 2025. How Humans Communicate Programming Tasks in Natural Language and Implications For End-User Programming with LLMs. In *CHI Conference on Human Factors in Computing Systems (CHI '25), April 26–May 01, 2025, Yokohama, Japan.* ACM, New York, NY, USA, 34 pages. https://doi.org/10.1145/3706598.3713271

1 Introduction

Large language models (LLMs) have been demonstrating remarkable jumps in performance in the past few years, resulting in their rapid application to a wide range of problems. One application area is program generation, where LLMs have been shown to be able to write *code* using only *natural language* prompts [7, 52, 58].

We imagine a future in which LLMs enable end users (individuals who may not have prior programming experience) to solve programming-like problems via a natural language interface, freeing them from needing to learn the syntax of traditional programming languages. To this end, we investigate the central issue of how humans communicate computational tasks using only natural language as an interface. Through two between-subjects online studies, we examine interface factors designed to support people expressing tasks in natural language, as well as attributes of the individuals themselves that contribute to success. To increase generalizability, we investigate the communication of tasks both between humans (Study 1) and from a human to an LLM (Study 1 and Study 2).

To be precise, we focus on **Information Processing Tasks (IPTs)**, which necessitate a calculation that involves a self-contained formal transformation of input to output. An example IPT is that "you will be given a list containing numbers. Report the sum of the positive elements." In our study, the communication of an IPT involves a **sender**, who substantially rephrases the IPTs we provide,

and a **receiver**, who interprets and acts on the sender's description. To measure understanding, we have both the sender and receiver answer **test cases**, specifying the output for provided inputs.

In preliminary work, we found that senders often struggled to convey tasks successfully and hypothesized that two design elements could be important for improving IPT communication. One element is the use of **examples**: the sender is asked to include appropriate input/output pairs for a given IPT when describing it in natural language to make their description more concrete. The other element is the opportunity to **interact**: the receiver is encouraged to send prose feedback to their sender if they felt that the sender's description was unclear in any way.

In our analysis, we distinguish between **non-programmers** and **programmers** (in the sense of having experience with traditional programming languages like Python) to understand whether the skills developed by programmers generalize to better understand and convey IPTs in natural language. We also qualitatively explore the characteristics of successful IPT communication.

Our data comes from two studies. **The goal of the first study** is to examine what experimental factors (programming experience, presence of examples, interaction affordances, IPT complexity) improve human-to-human IPT communication. We do this by pairing crowdworkers in real time and asking one member of the pair, the sender, to communicate an IPT to the other participant, the receiver. The receiver then answers test cases that serve as a proxy for how well the IPT was communicated. To determine what facets of human-to-human IPT communication translate to successful human-to-LLM communication, we also feed the sender's natural language IPT description to a collection of LLMs. In contrast, the **goal of our second study** is to investigate how humans communicate IPTs to LLMs *directly*. As such, we pair human senders with LLM receivers in Study 2. Because LLMs can either answer test cases directly or use the descriptions to produce code, we measure the performance of the LLM on both objectives across both studies.

The results from the two studies largely complement each other. For example, we find that individuals with programming experience tended to be more successful at communicating IPTs to both other humans and LLMs. However, we also find that the effect of programming experience was often complemented by, or secondary to the effects of, examples. However, programmers also exhibited certain different behaviors around examples (e.g., being more likely to send them) and interactivity (e.g., being less likely to interact) compared to non-programmers.

Finally, we investigate the types of interactions that are beneficial when communicating a task in English prose. In general, requests for clarification did not necessarily correlate with positive outcomes. We also observe significant differences in the way that LLMs and humans provide feedback. In particular, LLMs are much more likely to initiate feedback and to provide multiple points of feedback than humans. Further, LLMs occasionally generated feedback *types* that were not observed in humans, such as copyediting an IPT for clarity.

2 Related Work

To our knowledge, three prior papers discuss end-user programming using LLMs as a natural language interface. Two [27, 52] focus specifically on end-user programming through spreadsheet interfaces. In contrast, our study is focused on the *open-ended communication* of a *variety* of computational tasks to enable end-user programming. Similar to our work, Sarkar [50] posits that "generative AI will create qualitative and quantitative expansions in the traditional scope of end-user programming." This accurately summarizes the motivation of this paper; however, unlike the author we perform extensive experimentation to better understand *how* and *why* this might be achieved. Due to the dearth of directly related empirical work, we discuss related work on interfaces for end-user programming (Section 2.1) and code generating LLMs (Section 2.2).

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 [25, 34]. Many systems for end-user programming rely on graphical user interfaces and visual programming [33] rather than forcing users to memorize complex syntax. For instance, both the Blockly [14] and Scratch [44] languages let users express programs by connecting blocks and include fundamental programming concepts like loops and conditionals, but present them graphically [44]. Similarly, trigger-action programming gives users a graphical user interface for defining event-driven rules [17, 59] and has been popularized in recent years via commercial services like Zapier [43] and IFTTT [30, 60]. Current instantiations of trigger-action programming enable end users to define automations in domains including smart homes [59, 65, 67], robots [2, 24], social media [60], and workplaces [66]. Other systems, such as prototypes for creating web mashups [63], ask users to specify workflows graphically. More broadly, a recent literature survey [4] 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 [21] 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 [4], 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. We note that prior work on enabling natural language as an effective programming interface is largely complementary to this study, as we focus on IPTs more broadly rather than specific tasks. As such, strategies such as clearly mapping user intent to specific commands (and clearly providing feedback based on the affordances of the system) may be overly restrictive when one works with an LLM, as the space of successful inputs are much larger and the workloads more diverse [26, 31]. To accommodate this diversity of inputs, 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, aside from the small number of papers mentioned above, 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 [20].

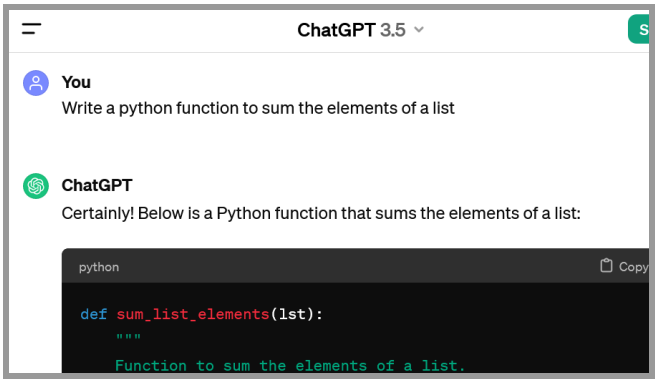

Figure 1: ChatGPT provides a chatbot-based interface that can be used to communicate IPTs.

Programming by demonstration (PbD) or by example has also long been studied as a type of end-user programming [33]. While not our main focus, we test a condition in which senders are encouraged to augment their natural language description with example test cases, essentially following the PbD paradigm.

2.2 Code Generators

This section discusses the technical origins of code generating LLMs (Sec 2.2.1), their usage as programming support tools (Sec 2.2.2), and their usage in the context of programming education (Sec 2.2.3). Hereafter, we use the term **code generators** to refer specifically to code generating LLMs.

2.2.1 Technical Origins. The relationship between code and natural language was brought to the forefront of research attention during the development of GPT-3 [5]. 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 [7]. Codex has since been incorporated into the backend of Github Copilot and iterated upon considerably. Other companies, such as Google and Meta, have similarly produced their own LLMs capable of high performance on coding tasks: Gemini and Code Llama, respectively [15, 48].

Modern LLMs are trained to produce the most likely next bit of text, or *token*, that should follow an input string [5, 15, 36, 42, 57, 62]. This enables humans to interact with, or **prompt**, LLMs by supplying them with input strings. Because code-generating LLMs are trained on both natural language and code, they can be prompted with *both* natural language and/or code. For example, a user may prompt the model, "Write a line of Python that checks if a number is even or odd", *or* "`is_even = .`" In either case, the model should produce an 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 as in the cases of ChatGPT (Figure 1) and Code Llama (Figure 2).

2.2.2 Code Generators as Programming Support Tools. Understanding LLMs as programming support tools has primarily been done

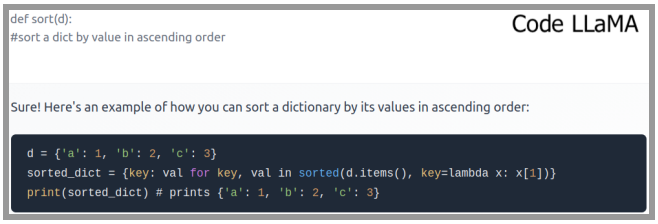

Figure 2: Meta Code Llama's interface is a widget that delineates between the prompt (top) and response (bottom).

through two lenses: characterizing the human-LLM interactions themselves or characterizing the correctness of produced code. Taking the former route, Lee et al. [22] develop an open-ended framework to understand human-LLM interactions across various tasks. However, many others adopt a more grounded approach towards understanding human-LLM interactions. Liu et al. [27], for example, develop a method of scoping natural language inputs 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. McNutt et al. [28] similarly map open-ended text to specific outcomes by characterizing the potential design space for notebook-based code assistants, noting that the most useful interfaces differ based on the task at hand. Similarly, Ferdowsi et al. [11] explore how an end-user inspection tool for LLM code in a spreadsheet might be designed in order to promote trust in the system. Others [18, 19, 29, 61] primarily investigate user perceptions as they work collaboratively with programming LLMs, with Ross et al. [46] particularly focusing on how different interfaces affect user satisfaction. One common theme among this work is that users' self-perception of productivity is often a misleading metric: users often do not work faster or produce correct code at a faster rate than control groups. This observation has critical implications for theoretical work advocating for self-perception as a metric for usability [56]. In contrast to these efforts to clearly scope LLM inputs to specific affordances, our work broadens the scope of inquiry. Namely, our study investigates the workloads themselves—our object of study is *any* self-contained computational task, which prevents us from necessarily tying the LLM's output to any specific system or affordance within.

Yet other work explores broad usability issues with prompting. Zamfirescu et al. [64] investigate LLM prompting by non-expert users, finding that when allowed to directly prompt models, users tended to over-generalize and prompted the models *opportunistically*, rather than *systematically*. Critically, Drosos et al. [9] find that even when researchers acted as a direct intermediary between primarily skilled, highly educated participants, those participants would *still* sometimes struggle to clearly articulate their queries. Difficulties of clearly forming intention are further theorized about in Sarkar's "intention is all you need" [51], which speculates that intention is a key prerequisite for using a GenAI tool. Our work expands on these broad ideas by performing a large scale empirical study specifically in the context of end-user programming (in contrast to Zamfirescu et al. [64]), and by providing evidence specifically in the domain of end-user programming to examine the theory that Sarkar [51] raised concurrently to this work.

Over-generalization re-emerged as a problem in the area of model prompting for code repair, as explored by Bajpai et al. [3], who found that they needed to precisely scope the model's feedback for effective debugging to occur; otherwise, the model would fail to precisely pinpoint the issue. Vaithilingam et al. [61] relatedly investigated usability aspects of code-producing LLMs, finding that users struggled to recover from incorrect or confusing code generated from the LLM. This echoes work in the security space investigating bugs—particularly security vulnerabilities—introduced by LLMs [38, 39, 49]. Those researchers generally find that the LLM does produce insecure code, yet the model does not introduce security vulnerabilities at a rate greater than humans [49]. While our work implicitly considers similar questions ("What if the LLM provides an output, such as feedback, that superficially appears correct and useful but actually misleads the end user?") our focus is not on subtle ways that the code produced may differ from users' expectations. We instead focus on factors that affect both human and LLM *understanding* of descriptions of a computational task.

Finally, natural language processing (NLP) researchers have broadly investigated LLM correctness during programming without human interaction. In contrast to self-perception, usability, or formal correctness metrics, a common NLP method of determining LLM code correctness is the notion of *functional correctness*, which states that code is correct if it passes associated unit tests [7, 15, 47, 48]. While model developers have primarily considered code correctness during model development, others have examined the quality of produced code afterwards [10, 32, 35, 55]. Our work is inspired by functional correctness and similarly utilizes test cases to check participant understanding and code correctness.

2.2.3 Code Generators and Programming Education. Some work has explored the application of code generators to programming education motivated by concerns that students may use the generators to complete their assignments [45]. Approaches have stemmed from evaluating code generators on introductory programming tasks to determine the extent to which they may be used either for cheating or as educational support tools [12, 13, 53]. 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, as the generator's abilities will likely improve over time.

3 Study 1 Methods

We first conducted an online study to understand human-to-human IPT communication in prose. We recruited participants of varying programming expertise and had them complete a survey. We then directed them to our custom-built web app, where they were paired with one another in real time. One member of the pair would then communicate an IPT to the other member. After each IPT communication, each member of the pair answered four test cases relevant to that IPT. Participants then completed another survey.

3.1 Recruitment and Demographic Survey

We recruited 242 participants from the Prolific crowdworking service [41]. 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. During recruitment, we reserved approximately half the slots on Prolific to users who self-reported programming experience. We compensated participants $8.00 USD either if they completed the study or if they were unable to complete the study due to no fault of their own. In the latter case, we compensated participants either if they were not paired with another crowdworker within 20 minutes (see Section 3.2) or if their assigned partner became unresponsive. Unresponsiveness was detected by our web app's interface. The criteria for being marked as unresponsive were either failing to move one's mouse or make a keystroke for ten continuous minutes or closing the study window and not rejoining within five minutes.

Our study was approved by the University of Chicago IRB. Participants were first directed to an initial survey with standard demographic questions (Appendix I.1). After completing the initial survey, they were redirected to our web app.

3.2 Pairing and Conditions

We first used our web application to pair participants with each other. Each participant stayed on a waiting screen until another participant was redirected to the same screen, at which point they were paired. The first member of the pair to land on the waiting screen was assigned the role of **sender**, who describes an IPT in prose. The second was assigned the role of **receiver**, who interprets the sender's rephrasing. Assigning the study roles in this way ensured that the sender was assigned in a round-robin manner based on the queue at the waiting screen.

We also wanted to explore the effects of various design elements, such as if adding an affordance by which receivers could ask questions about the IPT would help communication, as well as what role examples play in IPT communication. We used a between-subjects design to explore these design elements, assigning the experimental condition at the moment of pairing. Each pair of participants was assigned in a round-robin manner to either the **interactive** condition or **non-interactive condition**. In the interactive condition, the receiver could clarify the sender's IPT by pressing a button to open a chat window. This affordance was removed for the non-interactive condition. We also randomly assigned each pair to either the **examples** condition or **no-examples** condition. Five examples were presented to the sender in the examples condition, two of which captured edge cases. The sender was also asked to include their own examples in the IPT rephrasing. In the no-examples condition, the sender was not provided examples.

3.3 Primary Study Task

The main study task was for the sender to describe an IPT in prose to the receiver. From this, we could then investigate if the communication was successful. A key challenge we faced in designing our study was 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. This is because we hypothesized that the way that senders would structure their descriptions may have an effect on communication success. 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 investigate the benefits of programming experience. We thus decided to present the IPT to senders in prose and to omit any computer science jargon from the IPT. However, since the senders were also writing prose, we needed to ensure that they would not simply reiterate verbatim the prose IPT we provided them.

To address these challenges, we decided to give senders a *verbose* and *context-specific* prose description of each IPT in an interface element that prevented copy-pasting. We term this the **verbose description**. By verbose, we mean that we included sentences that provided no new information related to the fundamental computational task in the prose description. Similarly, by context-specific, we mean that the IPT itself is couched in a specific real-world situation rather than a pure mathematical representation. We hoped that both verbosity and the context-specific nature of these descriptions would both discourage copying our description (due to the quantity of text) and force participants to more deeply consider what the *fundamental computational task* was. We note that work conducted in parallel to ours also discussed verbosity as a technique [27]. Specifically, those researchers employed verbosity in conjunction with circumlocution, rather than our tactic of increasing context.

As an example of phrasing an IPT to be verbose and context-specific, consider the IPT of returning the maximum value in a list. The pure mathematical representation of this IPT could simply be "return the maximum value of some numbers." However, the *verbose description* of this IPT would add the real-world context of finding the maximum number of days late someone could pay their rent when comparing apartments. Similarly, we would include the irrelevant information of the specific city the individual was moving to, why they want to pay their rent late, and the fact that the apartments were all similar.

Because we write our IPTs in this manner, our instructions to senders asked them to rewrite the task more simply and more generally (i.e., not specific to a context), producing what we termed the **sender's rephrasing**. We gave participants the sample of rewriting "sort a list of milk brands…" as "sort a list of words" to clarify what it means to be general.

3.3.1 Selection of IPTs. We considered IPTs from a number of sources because no work to date has established which IPTs might be broadly representative of end-user programming tasks. The intuition here was to draw inspiration from how LLMs, experienced programmers, and beginner programmers had their programming competence measured. We specifically looked at four popular LLM benchmarks (HumanEval, APPS, MBPP, and BigBench datasets [1, 7, 16, 54]), a popular interview prep tool (LeetCode [23]), and an introductory programming course (MIT [6]). 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 [7]. We chose problems from HumanEval because they were largely nontrivial, self-contained, and did not require formal training to complete. However, HumanEval is a very popular LLM benchmark—pulling IPTs exclusively from HumanEval runs the risk of us testing IPT communication to LLMs for IPTs that the LLMs may have been trained on or otherwise optimized to correctly solve. To mitigate this, we developed half of our IPTs ourselves.

We selected IPTs of varying difficulty because more complex IPTs may be harder to communicate. On the simplest end, an IPT might be simply finding the maximum element in a list. At the other extreme, one may need to describe a custom math function. We then rephrased each IPT as described in Section 3.3 to create their **verbose description**. We finally note that while we did not consciously pull any IPTs from datasets other than HumanEval, some of our IPTs exist in other datasets due to their generic nature. An example of this is our eighth IPT, which asks if a string is a subsequence of another string [7, 23]. We believe this heavy overlap with problems in other testing suites is a good indicator that our selected IPTs are common programming subtasks. A summary of our chosen IPTs is shown in shown in Figure 14. The full IPT descriptions and their test cases are available in Appendix B.

3.4 Interface

After the sender and receiver are paired, the main study activities of communicating IPTs can occur. Critically, we do not distinguish between writing instructions to complete a task versus describing the task itself. This is intentional because we posit that *any* method of communicating an IPT that ultimately results in correct code or correct understanding is sufficient. Each participant was shown a view for communicating IPTs customized to their role and condition. The goal of the sender's view was to present the IPT they would have to rephrase and to give them the space to rephrase it. To this end, the sender's view (Figure 3) 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. Senders assigned to the examples condition were also given examples along with the verbose description. They were told to include their own examples when rephrasing the IPT.

Correspondingly, the goal of the receiver's view was to present them with the sender's rephrased IPT and potentially to allow them to respond to the rephrasing. To this end, the receiver's view (Figure 4) contained study instructions, a non-editable text box where the sender's rephrased IPT appeared, and a button to accept the rephrased IPT. The receiver could optionally iterate with the sender on the wording of the rephrasing in the *interactive* condition. A button to reject the sender's rephrased IPT was added in the interactive condition for this purpose. Rejecting the rephrased IPT opened a text box for the receiver to explain what they found unclear. A chat window (Figure 5) then opened for both the sender and receiver to optionally further discuss the receiver's request for clarification. Finally, we allowed the sender to optionally edit what they had written in response to the receiver's request. We make this optional in the case that the clarification is small enough that no changes to the sender's IPT rephrasing needs to be made (e.g., Receiver: "What test cases will I see?" Sender: "I don't know that information.") Once the sender finished making edits, they pressed a button to send the edited IPT back to the receiver. The receiver could again accept or reject the modified IPT, repeating this cycle of feedback as many times as the receiver felt necessary.

Once the receiver accepts the sender's IPT, we check for understanding on the part of both the sender and receiver. We do this by having each participant independently answer four **test cases** relevant to the IPT. For each test case, the participant supplies the expected output for an input. To ensure that participants are not ultimately quizzed on their memory, we provide the version of the

Table 1: Short descriptions of the 12 IPTs we tested in Study 1. Appendix B contains the full text given verbatim to senders.

Name	Short Summary of the IPT
max	Given a list of numbers, report the max.
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.
str_diff	Given two strings, report which letter has been added to the first string to arrive at the second.
exact_chg	Given three numbers (Small, Big, Goal), report T if Small times 1 plus Big times 5 equals Goal. 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 [].
num_even	Given a list of numbers, report how many even numbers are in the list.
is_subseq	Given two strings, report T if the first string is a subsequence of the other. Otherwise, report F.
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].
rmv_dup	Given a list, report the list after all instances of duplicates are removed, preserving the relative ordering.
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).

IPT that they saw on the previous screens. That is, the sender is provided the *verbose description* and the receiver is provided the sender's IPT rephrasing. We fix the test cases for each IPT. However, we randomize the order in which each participant sees them. To test both basic understanding as well as mastery, we test a variety of inputs. We formulate two of the test cases for each IPT to be **regular cases** which test common, standard scenarios. The other two cases are **edge cases** that test unusual scenarios. For example, for the IPT of finding the maximum of a list, a regular case may be the list "[1, 2, 3]" while an edge case could be "[0, 0, 0]."

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 [IPT] above."* After answering these questions, we consider the members of the pair to have communicated one IPT. We repeat the IPT communication process until each pair has communicated four IPTs, with the participant roles (sender and receiver) and condition held constant. We randomize which four IPTs that each pair communicates, as well as the order in which they see those IPTs.

3.5 Post-Task Survey

Our post-task survey (see Appendix I.2) focused on participants' self-perception, such as whether or not they perceived the study task to be difficult. We also use this survey to definitively classify each participant as a *programmer* or a *non-programmer* using three screening questions developed by Danilova et al. [8] and a fourth that simply asks the programming languages in which the participant is proficient. We count the fourth question as indicative of programming experience as long as the participant types *any* programming language. Participants who answer at least three questions correctly are considered programmers. We also included an attention check question; all participants answered it correctly.

3.6 Substituting LLMs For Human Receivers

We use data collected during our main study to perform an additional sub-study. This sub-study investigates if IPTs communicated between humans can be used verbatim for successful IPT communication to LLMs. Specifically, we took the final IPT phrasing (*post-interactivity* if interaction was permitted) and fed it to a variety of LLMs. We use the same test cases as we did for humans to measure LLM performance. However, unlike humans, LLMs can either directly answer test cases or produce code. We consequently

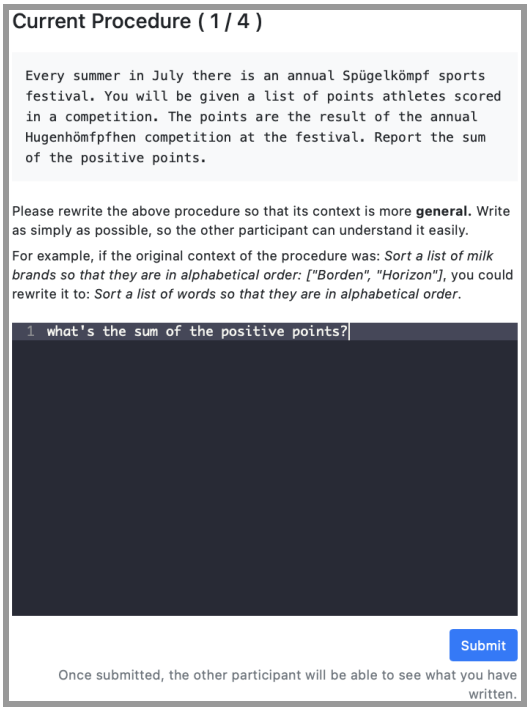

Figure 3: Sender's view in Study 1's non-interactive condition.

measure LLM performance for both possibilities, and prompt each LLM for each objective: once to **directly answer** test cases, and once to **produce code** in the form of a Python function that implements the IPT. The written code is then run to answer test cases. 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 (`gpt-4-0613`), Chat Llama (`Llama-2 Chat 70B hf`, `Llama-2 Chat 7B hf`), Code Llama (`CodeLlama 34B Instruct hf`, `CodeLlama 7B Instruct hf`), and Gemini (`Gemini Pro 1.0`) due to their state-of-the-art performance [15, 37, 48, 58]. 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.

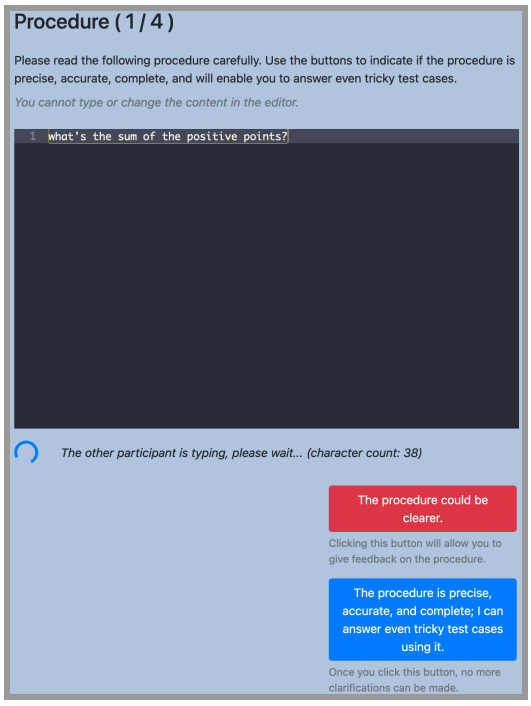

Figure 4: Receiver's view in Study 1's interactive condition.

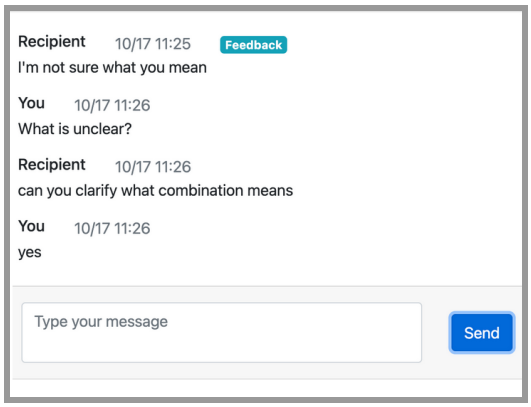

Figure 5: Chatbox shown if a receiver rejects the description.

3.6.1 LLM Code Correctness. We observe that there is no guarantee that an LLM will produce a correct function header. Namely, LLMs will sometimes produce a function that takes too few or too many arguments. Rather than indiscriminately mark these cases as incorrect, we first check if the IPT relies on its arguments being in a certain order. If so, we simply count the produced function as incorrect. However, the IPT is more or less agnostic to the order of its arguments, such as IPT `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 LLM's function header. For example, if the test case took two inputs (`x` and `y`) and the function header only took one input, we attempted to format the input as `[x, y]`. Finally, we also mark code that takes excessively long to execute (>10 seconds) or relies on input from the user as incorrect. 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 verified the results from code execution to ensure that the produced code legitimately attempts to solve the test cases instead of simply guessing.

3.7 Data Analysis

We use linear regression models for our quantitative analysis. These linear regressions enable us to report on each factor's impact on communication (e.g., the impact of age or the impact of being assigned to the **examples** condition) while controlling for other factors. The dependent variable (**DV**) is the number of test cases answered correctly by the sender, the receiver, or LLM. 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. 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 coefficients and p-values for all IVs. For our regression models, we merged smaller categories and always used the largest category as the baseline.

However, we sometimes 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.

We also perform qualitative analysis to assess the characteristics of sender's IPT rephrasings, examples, and interactions. Two researchers inductively developed a codebook following a thematic analysis approach. Rephrasings, examples, and interactions all had separate codebooks, 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 with under five occurrences with thematically similar codes.

To support both open science and potential follow-up work, we have publicly released de-identified versions of participants' descriptions in a GitHub repository [40].

4 Study 1 Results

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

4.1 Participants

Initially, 258 participants completed Study 1. 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" 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,

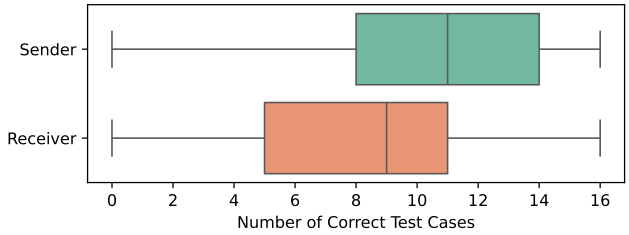

Figure 6: The distribution of how many of the 16 test cases each sender and receiver answered correctly.

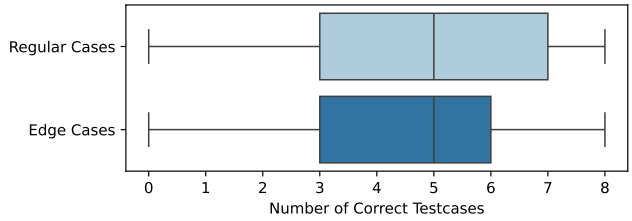

Figure 7: The distribution of how many of the 8 regular cases and 8 edge cases each participant answered correctly.

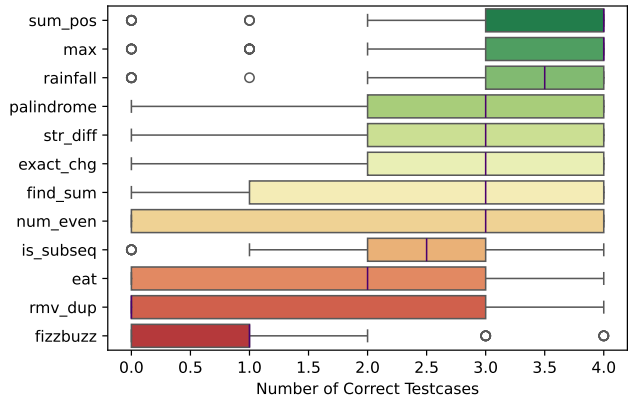

Figure 8: The distribution of how many of the 4 test cases for each IPT participants answered correctly.

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 a college degree, 34.3% held an associate's degree, 37.2% 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.

While we recruited our sample using Prolific's platform filters such that slightly over half of the participants reported prior programming experience and slightly less than half did not (Section 3.1), a number of self-reported programmers fared poorly on the standardized programming quiz [8] 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. Of the 72 programmers, 33 were senders while the other 39 were receivers.

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 difficulty of the IPT, 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 IPT rephrasing. As Figure 6 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. Participants did slightly better on regular test cases than edge cases, as shown in Figure 7. The x-axis is out of 8, as each participant encounters four IPTs, each with two edge cases. This difference was also significant (MWU, $p = .030$).

4.2.3 Performance By IPT. Participants' performance varied with IPT difficulty (Figure 8). Participants answered the most test cases correctly for `sum_pos` (mean: 3.1/4, median: 4) and the least for `fizzbuzz` (mean: 1.1, median: 1.0). From this, we speculate that **computationally simple IPTs are simple to communicate**. Further, we find some evidence that participants tended to **struggle with IPTs that inherit subtle English language ambiguities even if the IPT is computationally simple.** For example, the `num_even` IPT, which requires one to return the count of even numbers in a list, proved challenging for participants. Many unsuccessful participants returned *all even numbers* in the list, rather than the *count* of the even numbers in the list.

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 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. Note that *because the DV is defined in this way, any statistically significant effects are additive.* For example, if the sender is a programmer ($\beta = 0.45$) who is in the examples condition ($\beta = 0.60$), that sender is likely to get 1.05 more test cases correct on average than a non-programmer in the condition without examples.

Table 2 presents our regression results for senders answering test cases based on our verbose descriptions, while Table 3 presents the corresponding regression results for receivers answering test cases based on their paired sender's description. We note that the characteristics of the person rephrasing the IPT may impact the quality of the rephrasing. Correspondingly, the regression for receivers

Table 2: Linear regression whose dependent variable indicates how many of the four test cases *senders* answered correctly in Study 1 based on our verbose descriptions. Adjusted $R^2 = 0.302$.

Independent Variable	Baseline	β	SE	t	p
(Intercept)	–	3.266	0.250	13.063	**<.001**
Condition: Examples	Non-examples	0.598	0.121	4.951	**<.001**
Condition: Interactive	Non-interactive	0.097	0.120	0.813	.417
Sender is Programmer	Non-programmer	0.450	0.141	3.203	**.001**
Sender Age 18–24	25–34	0.086	0.207	0.416	.678
Sender Age 35–44	25–34	0.010	0.144	0.068	.945
Sender Age 45+	25–34	-0.682	0.168	-4.054	**<.001**
Sender is Non-male	Male	0.002	0.126	0.018	.986
Sender Degree: None	Bachelor's	-0.013	0.141	-0.096	.924
Sender Degree: Associate's	Bachelor's	0.076	0.211	0.361	.718
Sender Degree: Graduate	Bachelor's	0.309	0.173	1.790	.074
IPT: `max`	`sum_pos`	-0.197	0.272	-0.725	.469
IPT: `num_even`	`sum_pos`	-1.080	0.289	-3.742	**<.001**
IPT: `reignfall`	`sum_pos`	-0.117	0.306	-0.383	.702
IPT: `palindrome`	`sum_pos`	-0.616	0.279	-2.206	**.028**
IPT: `find_sum`	`sum_pos`	-1.036	0.289	-3.582	**<.001**
IPT: `rmv_dup`	`sum_pos`	-2.105	0.277	-7.607	**<.001**
IPT: `str_diff`	`sum_pos`	-0.791	0.280	-2.826	**.005**
IPT: `is_subseq`	`sum_pos`	-0.927	0.283	-3.280	**.001**
IPT: `exact_chg`	`sum_pos`	-0.659	0.275	-2.400	**.017**
IPT: `fizzbuzz`	`sum_pos`	-2.410	0.279	-8.629	**<.001**
IPT: `xyz`	`sum_pos`	-1.436	0.290	-4.957	**<.001**

also includes demographic IVs of the sender. Full demographic correlations with test case performance are shown in Appendix F.

Controlling for all other factors represented by IVs, we found that the number of test cases the receiver answered correctly was significantly correlated with the number of test cases the sender answered correctly ($\beta = 0.306$, $p < .001$). 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.** Further, echoing Section 4.2.3, we found that both senders and receivers answered fewer test cases correctly for five of the IPTs as compared to our baseline—`num_even`, `find_sum`, `rmv_dup`, `fizzbuzz`, and `xyz`—compared to our regression baseline, `max`.

4.4 Programmers vs. Non-Programmers

We found that senders with programming experience answered on average 0.45 more of the 4 test cases per IPT correctly than senders without programming experience ($\beta = 0.450$, $p < .001$). While **programming experience clearly helped understanding, its advantage was not overwhelming**, representing on average less than half a test case difference per IPT.

We observed a slightly different effect for receivers. Receivers with programming experience answered on average 0.25 more of the 4 test cases per IPT correctly than receivers without programming experience, and this effect was *not* statistically significant ($\beta = 0.254$, $p = .082$). **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 0.43 more of the 4 test cases per IPT correctly than receivers whose corresponding sender did not ($\beta = 0.433$, $p = .006$).

Table 3: Linear regression whose dependent variable indicates how many of the four test cases *receivers* answered correctly in Study 1 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). Adjusted $R^2 = 0.285$.

Independent Variable	Baseline	β	SE	t	p
(Intercept)	–	1.645	0.360	4.570	**<.001**
Condition: Examples	Non-examples	0.249	0.142	1.756	.080
Condition: Interactive	Non-interactive	0.061	0.129	0.473	.637
Receiver is Programmer	Non-programmer	0.254	0.146	1.743	.082
Receiver Age 18–24	25–34	-0.535	0.216	-2.473	**.014**
Receiver Age 35–44	25–34	-0.316	0.186	-1.699	.090
Receiver Age 44+	25–34	-0.391	0.169	-2.318	**.021**
Receiver is Non-male	Male	0.176	0.131	1.346	.179
Receiver Degree: None	Bachelor's	-0.014	0.171	-0.084	.933
Receiver Degree: Associate's	Bachelor's	0.132	0.210	0.627	.531
Receiver Degree: Graduate	Bachelor's	-0.281	0.187	-1.502	.134
Sender: # Test Cases Correct	–	0.306	0.050	6.104	**<.001**
Sender is Programmer	Non-programmer	0.433	0.155	2.786	**.006**
Sender Age 18–24	25–34	-0.701	0.242	-2.893	**.004**
Sender Age 35–44	25–34	-0.181	0.157	-1.149	.251
Sender Age 45+	25–34	-0.211	0.194	-1.083	.279
Sender is Non-male	Male	-0.226	0.139	-1.632	.103
Sender Degree: None	Bachelor's	0.255	0.159	1.608	.109
Sender Degree: Associate's	Bachelor's	0.260	0.227	1.146	.253
Sender Degree: Graduate	Bachelor's	0.477	0.191	2.499	**.013**
IPT: `max`	`sum_pos`	-0.038	0.287	-0.132	.895
IPT: `num_even`	`sum_pos`	-0.693	0.312	-2.225	**.027**
IPT: `reignfall`	`sum_pos`	-0.088	0.325	-0.271	.787
IPT: `palindrome`	`sum_pos`	-0.177	0.297	-0.595	.552
IPT: `find_sum`	`sum_pos`	-0.622	0.310	-2.006	**.045**
IPT: `rmv_dup`	`sum_pos`	-0.988	0.312	-3.168	**.002**
IPT: `str_diff`	`sum_pos`	-0.093	0.299	-0.310	.757
IPT: `is_subseq`	`sum_pos`	-0.268	0.303	-0.884	.377
IPT: `exact_chg`	`sum_pos`	-0.181	0.293	-0.617	.538
IPT: `fizzbuzz`	`sum_pos`	-1.030	0.321	-3.205	**.001**
IPT: `xyz`	`sum_pos`	-1.119	0.315	-3.555	**<.001**

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 compared to programmers. However, having programming experience affects both the individual's depth of understanding and ability to successfully communicate the IPT. Figure 9 plots these phenomena per participant (summing test cases across all 4 IPTs each participant saw). However, observing the difference in distributions is insufficient to understand *why* programmers do better. To understand what programmers do that end users do not, we look at both examples (Section 4.5) and interaction patterns (Section 4.6).

4.5 The Role of Examples

Senders who were shown examples alongside their verbose descriptions answered on average 0.60 more of the 4 test cases per IPT correctly than senders not shown examples ($\beta = 0.598$, $p < 0.001$). However, we note that only 32 of the 48 senders (66.7%) randomly assigned to our examples condition (in which they were asked to provide examples to the receiver) actually did so. Because all senders who were assigned to the examples condition were shown examples, but only two-thirds of them *sent* examples despite

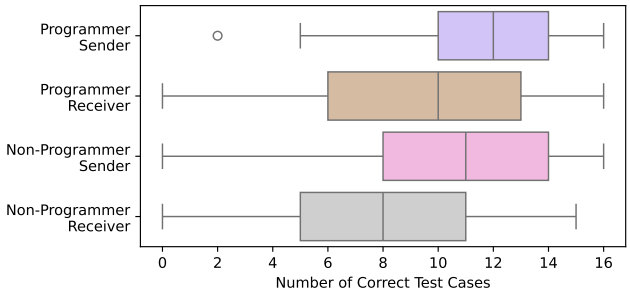

Figure 9: The distribution of how many of the 16 test cases (across IPTs) each participant answered correctly based on role and programming experience.

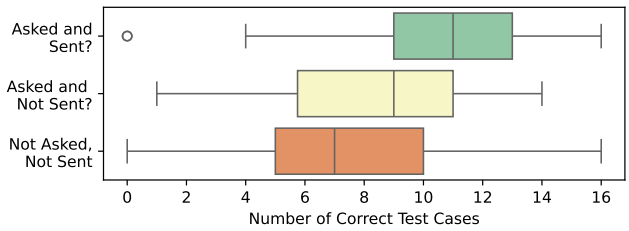

Figure 10: Receiver performance based on whether the sender included examples. Only one pair sent examples when they were not asked to; that pair is not shown.

all passing our study's attention check, we speculate that examples may simply be hard for individuals to formulate.

Whether the sender chose to send examples to their corresponding receiver was not significantly correlated with how many test cases their receiver answered correctly. Figure 10 plots receiver performance when their corresponding sender included one or more examples in their IPT rephrasing, summing across all four IPTs each participant saw. While sending examples was not significant, we found that the sender *reusing* one or more examples *was* significant. On average, the receiver got 0.75 more test cases *wrong* when the sender reused one or more examples ($\beta = -0.750$, $p = 0.047$). We include the full regressions over examples in Appendix H.2.

4.5.1 Do programmers and non-programmers have different behavior around examples? We first examine trends relating to the quantity of examples. We found that programmers sent examples at a greater rate when requested. Specifically, when looking at the rephrasings of those assigned to the examples condition, 66.1% (37/56) of programmers' IPT descriptions contained examples, compared to 48.5% (66/136) of non-programmers' IPT descriptions. When programmers and non-programmers sent examples, they ultimately sent a similar number of examples: 2.73 examples on average for programmers versus 2.75 for non-programmers. However, when they did so, there was greater *variance* in the quantity of examples that non-programmers sent (standard deviation: 1.65) as compared to programmers (standard deviation: 1.24). Further, the *median* number of examples sent was higher for programmers (median: 3) than non-programmers (median: 2). Curiously, we also found that programmers tended to write IPTs with fewer entirely correct examples (14/37, 37.8%) than non-programmers (28/66, 42.3%). We speculate that this slight disparity may be due to the fact that programmers wrote a greater median number of examples per IPT and were slightly more likely to include examples covering edge cases than non-programmers (28.6% vs 21.3% of IPTs with examples).

Regarding the qualities of those examples, we found that programmers sent more legible examples. Specifically, programmers were more likely to clearly format their examples such that each example's **inputs and output were clear**: 50.0% (28/56) of programmers' IPTs had examples clearly formatted, compared to only 32.4% (44/136) of non-programmers' IPTs. Example reuse was slightly more common for non-programmers. Specifically, 11.8% of non-programmers (16/136) **reused** at least one example that we provided to them in our verbose description, while 8.9% of programmers (5/56) reused at least one example. Similarly, when they included examples, non-programmers were more likely to have reused *every* example from what we provided: 5.9% (8/136) of non-programmer IPTs had entirely reused examples, while only one programmer had an IPT with entirely reused examples (1.8%, 1/56). Finally, we remark that both programmers and non-programmers included explanations for the answers in their examples at roughly equal rates: 14.3% of programmers' IPTs (8/56) included explanations and 13.2% of non-programmers' IPTs (18/136) included explanations.

We speculate that these results suggest that it is easier for programmers to create and express examples than non-programmers, and they might intuitively understand that being able to clearly delineate inputs and outputs is important when communicating examples. However, neither group appears to be more inherently predisposed to include explanations along with their examples.

4.6 The Impact of Interactivity

While we hypothesized that interactivity would help communication, we did not observe this to be the case.

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. Further, pairs with at least one programmer were 12.2% *less* likely to interact than pairs with no programmers: pairs with one programmer interacted 65.6% of the time, while pairs with no programmers interacted 77.8% of the time.

Interestingly, **interaction did not appear to improve outcomes for the receiver.** Receivers who did *not* interact got on average 0.47 more test cases correct than their senders ($\beta = 0.473$, $p = 0.42$). We hypothesize this may be because receivers were more likely to interact when the sender's description was unclear.

4.6.1 What is a useful interaction? Because we observed interaction to have minimal impact on the receiver's ability to answer test cases, we qualitatively analyzed the interactions that did occur to understand why. Many interactions 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 (Cohen's $\kappa = 0.77$). 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);

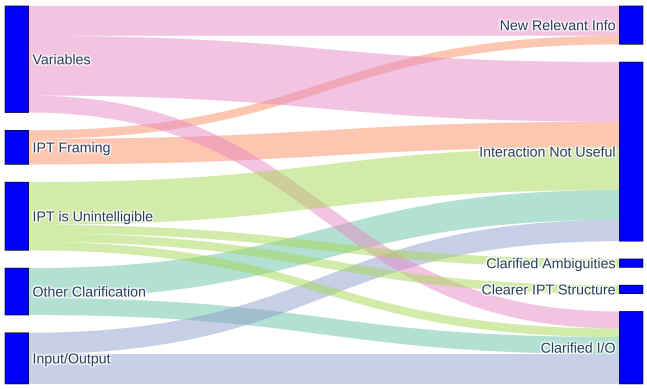

Figure 11: A sankey diagram connecting themes of receivers' clarification requests about senders' IPT descriptions and their associated outcomes. For visual clarity, we dropped all request-outcome pairs that only occurred once.

the sender responds with *new, relevant information*; the interaction *clarifies the IPT's inputs and/or output*; or the sender and receiver 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, 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, a participant wrote, "It seems very wordy and confusing. I still don't know what two numbers [...] to find or how I am supposed to do that." Finally, 13 requests (15.9%) did not neatly fall under the other categories, including grammatical edits or confusion over math symbols (e.g., confirming that '>' means "greater than").

Figure 11 summarizes how these requests mapped to outcomes. **Requests to clarify variables and I/O appear the most beneficial to improving IPT communication**. These requests led to positive outcomes 48.1% and 61.5% of the time, respectively.

4.7 Communication to LLMs

Recall that we are ultimately focused on the possibility of LLMs being an effective natural-language programming interface for end users. To this end, we investigate if the IPT descriptions communicated between humans can also successfully communicate IPTs to LLMs. To this end, 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. We then measured each LLM's ability to answer test cases both when answering directly and when producing Python code to do so (see Section 3.6). We found that all LLMs except Gemini were more adept at answering test cases directly, rather than producing code. With the exception again of Gemini, models' relative performance when directly answering test cases correlated strongly with their ability to write code.

Table 4 compares the performance of these LLMs against each other and against the human receivers. We observe that GPT-4, Gemini, and Code Llama 34B demonstrated the best performance, so we restrict our discussion to those models. GPT-4 performed the most comparably to human receivers in aggregate, slightly outperforming them. GPT-4 answered 59.8% of test cases correctly, while human receivers answered 58.9% correctly. Furthermore, GPT-4 produced code that passed all test cases 36.0% of the time.

Table 4: Average performance of all tested LLMs and humans. "Direct Answering" indicates that the LLM directly responded to the test case, while "Code" indicates that the LLM produced a Python function which was then run on test cases.

	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 5: Linear regression with the dependent variable indicating how many of the four test cases *GPT-4* answered correctly in Study 1 when *directly answering*. Adjusted $R^2 = 0.292$.

Independent Variable	**Baseline**	β	**SE**	**t**	**p**
(Intercept)	–	2.444	0.292	8.374	**<.001**
Condition: Examples	Non-examples	0.186	0.124	1.501	.134
Condition: Interactive	Non-interactive	-0.101	0.120	-0.841	.401
Sender: # Test Cases Correct	–	0.327	0.046	7.042	**<.001**
Sender is Programmer	Non-programmer	0.152	0.142	1.070	.285
Sender Age 18–24	25–34	-0.025	0.207	-0.120	.905
Sender Age 35–44	25–34	-0.104	0.144	-0.720	.472
Sender Age 45+	25–34	-0.319	0.171	-1.869	.062
Sender is Non-male	Male	-0.122	0.126	-0.968	.334
Sender Degree: None	Bachelor's	0.096	0.141	0.684	.494
Sender Degree: Associate's	Bachelor's	0.152	0.210	0.722	.471
Sender Degree: Graduate	Bachelor's	0.100	0.173	0.581	.561
IPT: `max`	`sum_pos`	-0.810	0.271	-2.986	**.003**
IPT: `num_even`	`sum_pos`	-0.997	0.292	-3.411	**<.001**
IPT: `reignfall`	`sum_pos`	-0.956	0.305	-3.131	**.002**
IPT: `palindrome`	`sum_pos`	-0.359	0.280	-1.280	.201
IPT: `find_sum`	`sum_pos`	-0.893	0.293	-3.053	**.002**
IPT: `rmv_dup`	`sum_pos`	-1.576	0.293	-5.380	**<.001**
IPT: `str_diff`	`sum_pos`	-0.885	0.282	-3.142	**.002**
IPT: `is_subseq`	`sum_pos`	-0.521	0.285	-1.826	.069
IPT: `exact_chg`	`sum_pos`	-0.666	0.276	-2.416	**.016**
IPT: `fizzbuzz`	`sum_pos`	-1.580	0.300	-5.262	**<.001**
IPT: `xyz`	`sum_pos`	-1.449	0.297	-4.887	**<.001**

Table 6: Linear regression with the dependent variable indicating how many of the four test cases *GPT-4* answered correctly in Study 1 when *producing Python code*. Adjusted $R^2 = 0.301$.

Independent Variable	Baseline	β	SE	t	p
(Intercept)	–	2.363	0.337	7.022	**<.001**
Condition: Examples	Non-examples	0.187	0.143	1.314	.190
Condition: Interactive	Non-interactive	-0.087	0.138	-0.634	.526
Sender: # Test Cases Correct	–	0.332	0.054	6.202	**<.001**
Sender is Programmer	Non-programmer	0.244	0.163	1.494	.136
Sender Age 18–24	25–34	0.244	0.239	1.024	.306
Sender Age 35–44	25–34	-0.074	0.166	-0.445	.657
Sender Age 45+	25–34	-0.688	0.197	-3.497	**<.001**
Sender is Non-male	Male	-0.220	0.145	-1.516	.130
Sender Degree: None	Bachelor's	0.108	0.162	0.667	.505
Sender Degree: Associate's	Bachelor's	0.280	0.242	1.153	.249
Sender Degree: Graduate	Bachelor's	0.369	0.199	1.853	.065
IPT: `max`	`sum_pos`	-1.757	0.313	-5.619	**<.001**
IPT: `num_even`	`sum_pos`	-0.395	0.337	-1.171	.242
IPT: `reignfall`	`sum_pos`	-0.912	0.352	-2.590	**.010**
IPT: `palindrome`	`sum_pos`	0.105	0.323	0.327	.744
IPT: `find_sum`	`sum_pos`	-1.446	0.337	-4.287	**<.001**
IPT: `rmv_dup`	`sum_pos`	-1.809	0.338	-5.356	**<.001**
IPT: `str_diff`	`sum_pos`	-1.422	0.325	-4.378	**<.001**
IPT: `is_subseq`	`sum_pos`	-1.353	0.329	-4.117	**<.001**
IPT: `exact_chg`	`sum_pos`	-1.351	0.318	-4.251	**<.001**
IPT: `fizzbuzz`	`sum_pos`	-1.651	0.346	-4.769	**<.001**
IPT: `xyz`	`sum_pos`	-1.242	0.342	-3.632	**<.001**

Parallel to our regression models for humans, we also built regression models for the LLMs. The body of the paper presents those models for GPT-4, specifically when GPT-4 answered test cases directly (Table 5) and when GPT-4 wrote code (Table 6). The appendix contains parallel tables for Code Llama (Tables 15–16) and Gemini (Tables 17–18). From these regressions, we compare and contrast LLM behavior to human receivers.

Models seemed to struggle with similar tasks as humans: human receivers, senders, and all models struggled with `find_sum`, `rmv_dup`, `fizzbuzz`, and `xyz`. Notably, the sender being a programmer only had a statistically significant effect for Code Llama 34B ($\beta = 0.443$ when producing code) and Gemini ($\beta = 0.317$) when producing code to answer test cases. The sender being a programmer was *not ever* statistically significant when the model was directly answering test cases.

4.8 Study 1: Should LLMs Have Failed?

A natural question is whether an LLM *should* have answered the test cases correctly given a sender's rephrased description. Answering this question provides a sense of whether more advanced LLMs would likely be able to answer those test cases correctly, or whether the sender's description was flawed. To this end, we qualitatively code a random sample of 100 IPT descriptions from senders. Of those descriptions, we judged that 45 contained all information needed to solve both regular and edge cases. For those, the senders answered 3.53/4 test cases correctly on average. GPT-4 performed well in these cases, answering 3.24/4 test cases correctly when directly answering and 3.47/4 test cases correctly when producing code. However, **33 of the 100 IPT descriptions contained none of the necessary information** to answer either regular or edge cases for that IPT. GPT-4 performed surprisingly well given this situation: 1.85/4 test cases were correct when directly answering, and 1.27/4 when producing code. We further note that 9/100 descriptions included no instructions on how to complete the IPT, while 11/100 were deemed incomprehensible by our expert coders. An example of an IPT that was coded as both having no instructions and as being incomprehensible is: "input-0-2-4-6-8 [line break] 2-4-6-8."

We then examined other qualitative features of the descriptions. We found a relatively even split between senders remembering to include the requisite information necessary to answer **all edge cases** (48/100) versus including **none** of that information (43/100). Similarly, about a quarter of the descriptions (26/100) failed to clearly define some information that was integral to the IPT. An example of a description **missing definitions** is illustrated by the statement, "Given a number of dollars, in the values Small and Big, determine if you can provide an amount exactly equal to Goal." The sender failed to mention that "Small" is the number of $1 bills and "Big" is the number of $5 bills. Relatedly, it was fairly common for individuals to accidentally convey an ultimately different IPT than the one we intended. For example, rather than convey "Report T *if it is possible to* obtain the exact amount needed given some quantity of $1 and $5 bills", one sender accidentally communicated "[Report T] *if you have* the exact amount needed." We find that almost a third (31/100) of descriptions ultimately conveyed a related, yet incorrect, IPT. Further, 18/100 descriptions contained information extraneous to the IPT, and this unsurprisingly seemed to correlate with poor understanding on the part of the sender, who answered 1.72/4 test cases correctly on average for those descriptions. Conversely, senders who wrote descriptions that had clear formatting (16/100) tended to average 3.94/4 test cases correctly. We note that 13/100 descriptions were written in a procedural, step-by-step format. Curiously, GPT-4 seemed to do worse when producing code for these descriptions than when directly answering (2.38/4 vs. 3.08/4 test cases correct, respectively). Senders who wrote these step-by-step descriptions seemed to understand the IPT, answering on average 3.46/4 test cases correctly themselves.

4.9 Limitations

We note two key classes of limitations for our results. While our IPTs were informed by common programming tasks, it is unclear how representative they are of end-user workloads either now or in a possible future in which LLMs have become a common interface for end-user programming. Further, we phrased IPTs specifically to study IPT communication (see Section 3.3), which is unlikely to be representative of how end users will have IPTs communicated to them. In this vein, our examples (when present) may not have encouraged participants to broadly consider the nature of the IPT, instead causing them to focus on those specific inputs and outputs.

Regarding LLMs, we consider that humans might structure information in a different way when communicating directly with an LLM than they would with another human. Further, the LLMs we query may have been trained on similar IPTs due to how common they are (e.g., finding the maximum element of a list may be a subroutine for data analysis or part of a sorting algorithm). End users with uncommon programming workloads may experience different behaviors when querying LLMs than what we report here.

5 Study 2 Methods

Our insights from Study 1 focused on how humans communicate IPTs to other humans and how LLMs interpret them. However, we did not investigate how humans communicate IPTs directly to LLMs. Our second study addresses this limitation and others by making three methodological changes. First, we construct our Study 2 IPTs to be more representative of how end-users *might in the future* leverage LLMs to perform computation in their daily life. Further, we decided to communicate IPTs to the human sender primarily through a visual depiction because the domain-specific nature of the new IPTs prevented us from asking participants to generalize them. Finally, to better understand interaction for language models, we replaced the human receiver with the best-performing LLM from Study 1, GPT-4. Unless otherwise specified, the other methods remained the same as in Study 1.

5.1 IPT Selection

We collaboratively developed five IPTs that we believe end users might want to use an LLM to solve. Short descriptions of these IPTs are listed in Table 7, while their corresponding full versions are listed in Appendix D. We also changed the fundamental way in which we presented IPTs from text to a predominantly visual format. Notably, we could not ask senders to change the context in which an IPT occurs because one of our main goals was to have the IPTs reflect actual workloads in which end users might be interested. We therefore used images to ensure that senders would not copy or paraphrase text we wrote. As in Study 1's text descriptions, we include distractor elements in the images. An example of a visual IPT is shown in Figure 12; the Phryges are distractors.

We also wanted to make our IPTs more computationally complex after Study 1 found that straightforward IPTs like "find the maximum element of a list" were trivial to communicate. In particular, every IPT now involved multiple steps of reasoning and had at least one edge case built in. For example, for the `surcharges` IPT, the surcharges for all items is 20% of the items cost unless that person is a member—in that case, there is only a $1 surcharge on each item. We also made the change that *all senders* were now presented with examples to ensure understanding despite the increased complexity. We presented each sender with three examples, two of which captured regular behavior and one which demonstrated what to do during an edge case. To ensure that senders actually sent examples when assigned to the examples experimental condition, we included a separate text area for writing the examples and would not let those participants proceed if that area was blank.

5.2 Pairing

A motivating factor of Study 2 was to determine how humans communicate IPTs directly to LLMs. Accordingly, all participants in Study 2 were senders and were *paired with an LLM as the receiver*. We also changed the interaction condition to accommodate the LLM. Specifically, we only allowed the LLM to provide feedback once per IPT. This design prevents cases where the LLM essentially traps the sender in the study by constantly stating that their IPT rephrasing is unclear. To encourage the LLM to provide useful feedback during interaction, we used the following system message as part of its prompt: "Below are instructions to solve a computational task. If the instructions address what to do in all possible contexts, even tricky test contexts, write 'understood.' Otherwise, if the description could be clearer, provide succinct feedback." To produce code, we lightly copyedit the prompt from our main study for clarity to "Write a single Python function which implements the process described above." For direct answering, we prompt the LLM, "Please determine the correct answer to the test context using the instructions above."

Figure 12: The `bill` IPT, which is emblematic of tasks requiring arithmetic that is straightforward, yet cumbersome, for a human.

6 Study 2 Results

In this section, we describe the results of having human senders communicate IPTs to GPT-4.

6.1 Participants

Initially, 101 participants completed Study 2. We excluded six, five of whom seemed to use ChatGPT to generate descriptions and one who did not answer the majority of study questions, leaving 95 participants as our sample. Among these 95 participants, 39.6% did not have a college degree, 10.4% held an associate's degree, 37.9% held a bachelor's degree, and 11.6% held a graduate degree. In terms of age, 17.9% were 18–24 years old, 36.9% were 25–34 years old, 24.2% were 35–44 years old, and the remaining 21.1% were 45 years old or older. Finally, 46.3% of participants identified as male, 51.6% identified as female, and 2.1% identified as non-binary/third gender. We categorized 33.7% of participants as programmers and the rest as non-programmers based on their answers to the same questions as in Study 1. Our regression baselines were analogous to Study 1.

6.2 Overall Correctness

Recall that each pair was randomly assigned four IPTs with four test cases each. Across their assigned IPTs, the human senders averaged 8.7/16 test cases correct (54.2%), with a median of 9/16 (56.3%) and standard deviation of 4.1. When directly answering test cases, GPT-4 averaged 6.4/16 test cases correct (39.7%), with a median of 6/16 (37.5%) and standard deviation of 3.1. However, when generating Python code, GPT-4 did less well, averaging 2.8/16 test cases correct (17.6%), with a median of 2/16 (12.5%) and standard deviation of 2.4. At the per-IPT level when directly answering test cases, GPT-4 answered more test cases correctly than the sender 48.7% of the time, GPT-4 and the sender answered the same number of test cases correctly 35.3% of the time, and GPT-4 answered more test cases

Table 7: Succinct descriptions of the 5 IPTs we tested in Study 2.

Name	Short Summary of the IPT
`bill`	Given the prices of some items and special promotions, determine the total bill.
`surcharges`	Given items in a cart and special rules (based on who is a member, if a person brought a reusable bag) determine the total surcharges imposed.
`format`	Given some out-of-date data, update the format of the data (e.g., from MM/DD/YYYY to DD/MM/YY).
`scheduling`	Given some room requests and rules about when rooms can be shared, determine what the room schedule will look like.
`iot`	Given three IOT devices and the rules about when they will turn on or off, determine which devices are on or off.

Table 8: Linear regression with the dependent variable indicating how many of the four test cases *GPT-4* answered correctly in Study 2 when *directly answering*. Adjusted $R^2 = 0.378$.

Independent Variable	Baseline	β	SE	t	p
(Intercept)	–	0.973	0.191	5.100	**<.001**
Condition: Examples	Non-examples	-0.005	0.109	-0.043	.966
Condition: Interactive	Non-interactive	0.198	0.102	1.945	.053
Sender: # Test Cases Correct	–	0.371	0.037	9.957	**<.001**
Sender is Programmer	Non-programmer	0.384	0.112	3.435	**<.001**
Sender Age 18–24	25–34	0.123	0.148	0.832	.406
Sender Age 35–44	25–34	-0.183	0.131	-1.391	.165
Sender Age 44+	25–34	0.153	0.145	1.058	.291
Sender is Non-male	Male	-0.048	0.105	-0.452	.652
Sender Degree: None	Bachelor's	-0.078	0.114	-0.688	.492
Sender Degree: Associate's	Bachelor's	0.318	0.182	1.742	.082
Sender Degree: Graduate	Bachelor's	0.161	0.166	0.971	.332
IPT: `surcharges`	`bill`	-0.740	0.161	-4.595	**<.001**
IPT: `format`	`bill`	-0.828	0.163	-5.095	**<.001**
IPT: `scheduling`	`bill`	-0.513	0.160	-3.215	**.001**
IPT: `iot`	`bill`	0.151	0.160	0.946	.345

Table 9: Linear regression with the dependent variable indicating how many of the four test cases *GPT-4* answered correctly in Study 2 when *producing Python code*. Adjusted $R^2 = 0.274$.

Independent Variable	Baseline	β	SE	t	p
(Intercept)	–	0.893	0.197	4.540	**<.001**
Condition: Examples	Non-examples	0.419	0.112	3.738	**<.001**
Condition: Interactive	Non-interactive	0.029	0.105	0.279	.781
Sender: # Test Cases Correct	–	0.189	0.038	4.905	**<.001**
Sender is Programmer	Non-programmer	0.286	0.115	2.481	**.014**
Sender Age 18–24	25–34	-0.218	0.152	-1.431	.153
Sender Age 35–44	25–34	-0.166	0.136	-1.224	.222
Sender Age 44+	25–34	-0.147	0.149	-0.988	.324
Sender is Non-male	Male	0.084	0.109	0.776	.438
Sender Degree: None	Bachelor's	-0.117	0.117	-0.992	.322
Sender Degree: Associate's	Bachelor's	-0.068	0.188	-0.362	.718
Sender Degree: Graduate	Bachelor's	-0.156	0.171	-0.909	.364
IPT: `surcharges`	`bill`	-0.699	0.166	-4.207	**<.001**
IPT: `format`	`bill`	-1.215	0.168	-7.248	**<.001**
IPT: `scheduling`	`bill`	-1.141	0.165	-6.929	**<.001**
IPT: `iot`	`bill`	-0.418	0.165	-2.534	**.012**

correctly than the sender 16.1% of the time. When writing Python code, GPT-4 answered more test cases correctly than the sender 5.0% of the time, GPT-4 and the sender answered the same number of test cases correctly 26.3% of the time, and the sender answered more test cases correctly 68.7% of the time.

We now focus on GPT-4 directly answering test cases. Table 8 reveals that neither the sender being assigned to the examples condition nor to the interactive condition had any significant effect on GPT-4's ability to directly answer test cases. However, parallel to Study 1, the number of test cases the corresponding sender answered correctly was a significant predictor of how many test cases GPT-4 answered correctly ($\beta = 0.371$, $p < 0.001$), as was the sender's programming experience ($\beta = 0.384$, $p < 0.001$).

When GPT-4 produced code, the number of test cases the sender answered correctly ($\beta = 0.189$, $p < 0.001$) and the sender having programming experience ($\beta = 0.286$, $p = 0.014$) again correlated with GPT-4 answering more test cases correctly, as shown in Table 9. However, unlike for direct answering, the sender being assigned to the examples condition had a significant positive effect when GPT-4 produced code ($\beta = 0.419$, $p < .001$).

Table 10: Linear regression correlating our qualitative codes of the descriptions written by participants in the *examples* condition and how many of the four test cases *GPT-4* answered correctly in Study 2 when *directly answering*. Adjusted $R^2 = 0.281$.

Independent Variable	Baseline	β	SE	t	p
(Intercept)	–	1.377	0.341	4.038	**<.001**
# of Examples	–	0.135	0.088	1.532	.128
Correctness: Mixed Correctness	All Correct	-0.434	0.259	-1.673	.097
Correctness: All Incorrect	All Correct	-0.929	0.258	-3.609	**<.001**
Correctness: No Examples	All Correct	-1.195	0.466	-2.565	**.011**
Reused Our Examples	Did Not	0.327	0.394	0.828	.409
Properly Formatted	Not	0.498	0.264	1.884	.062

6.3 The Lexicon Used to Rephrase IPTs

Programmers and non-programmers used different language when describing IPTs. We examined the unique words each participant used across IPT rephrasings, then analyzed the frequencies of words across participants. We observed that programmers tended to use words reminiscent of control flow—"otherwise," "change," "list," "output," and "follow." In contrast, non-programmers were more likely to use words where control flow was implicit, including "answer," "format," and "determine." Programmers also tended to use precise words like "multiply," "combine," "subtract," "sorted," and "state."

6.4 Characteristics of Examples

To gauge what types of examples are beneficial for communicating IPTs to GPT-4, we use both qualitative and quantitative analysis.

6.4.1 Comparison of programmers and non-programmers regarding examples when communicating to an LLM. As in Section 4.5.1, we first examine trends in the quantity of examples. Although we changed our user interface in Study 2 to more strongly nudge participants to send examples, we ultimately observed differences in the rates at which programmers and non-programmers sent examples. Every programmer's IPT rephrasing except for one (39/40, 97.5%) contained at least one example. In contrast, non-programmers sent

Table 11: Linear regression correlating our qualitative codes of the descriptions written by participants in the *examples* condition and how many of the four test cases *GPT-4* answered correctly in Study 2 when *writing Python code*. Adjusted $R^2 = 0.078$.

Independent Variable	Baseline	β	SE	t	p
(Intercept)	–	0.828	0.413	2.004	**.047**
# of Examples	–	0.073	0.107	0.683	.496
Correctness: Mixed Correctness	All Correct	-0.246	0.314	-0.784	.434
Correctness: All Incorrect	All Correct	-0.675	0.312	-2.165	**.032**
Correctness: No Examples	All Correct	-0.737	0.565	-1.306	.194
Reused Our Examples	Did Not	-0.083	0.478	-0.174	.862
Properly Formatted	Not	0.354	0.320	1.108	.270

Table 12: Linear regression correlating our qualitative codes of the descriptions written by participants in the *interactive* condition and how many of the four test cases *GPT-4* answered correctly in Study 2 when *directly answering*. Adjusted $R^2 = 0.115$.

Independent Variable	Baseline	β	SE	t	p
(Intercept)	–	1.900	0.214	8.896	**<.001**
Interacted: No	Yes	0.188	0.295	0.638	.525
# Useful Requests	–	-0.306	0.077	-3.995	**<.001**
# Irrelevant Requests	–	-0.106	0.096	-1.108	.269
# Useful Responses	–	0.309	0.108	2.853	**.005**
# Ineffective Responses	–	-0.007	0.134	-0.051	.960

Table 13: Linear regression correlating our qualitative codes of the descriptions written by participants in the *interactive* condition and how many of the four test cases *GPT-4* answered correctly in Study 2 when *writing Python code*. Adjusted $R^2 = 0.014$.

Independent Variable	Baseline	β	SE	t	p
(Intercept)	–	0.884	0.205	4.323	**<.001**
Interacted: No	Yes	0.028	0.283	0.098	.922
# Useful Requests	–	-0.164	0.073	-2.244	**.026**
# Irrelevant Requests	–	-0.077	0.092	-0.844	.400
# Useful Responses	–	0.147	0.104	1.418	.158
# Ineffective Responses	–	0.031	0.129	0.238	.812

at least one example in 89.6% of IPT descriptions (86/96). While, as in Study 1, the median number of examples sent by programmers tended to be higher (programmer median: 2.5, non-programmer median: 1), the average number of examples sent by these groups diverged, which we did not observe in Study 1. Specifically, programmers sent on average 2.5 examples per IPT, while non-programmers sent on average 1.8 examples per IPT. Programmers reused at least one example 5.0% of the time (2/40), while non-programmers reused at least one example 6.3% of the time (6/96). Consistent with the results of Study 1, we also observed differences across groups in the clarity of examples' inputs and output, as well as the number of examples sent. Programmers clearly formatted the IPT's inputs and outputs 92.5% of the time (37/40), while non-programmers did so 71.9% of the time (69/96). Programmers had an average of 2.1 correct examples per IPT, while non-programmers had an average of 1.1 correct examples per IPT. Relatedly, programmers included at least one incorrect example 37.5% of the time (15/40), while non-programmers did so 50.0% of the time (48/96).

From these results, we conclude that while our UI change did increase the rate at which both programmers and non-programmers sent examples, programmers were still more likely to include more examples, as well as higher quality examples. These trends seemed to hold independent of both IPT complexity and whether the receiver was an LLM or human.

6.4.2 Quantitative Effects of Examples. We examine the quantitative effect(s) of examples through the lens of our linear regression models (Tables 10 and 11). We note that entirely incorrect examples had a significant effect regardless of if GPT-4 was producing code or directly answering test cases. Further, GPT-4 seemed more vulnerable to a *lack* of examples when direct answering ($\beta = -1.195$, $p = .011$) rather than when producing code ($\beta = -0.737$, $p = .194$). We speculate this may be due to LLMs having being optimized for n-shot learning. In other words, LLMs are trained to modify their behavior to be consistent with any examples provided.

6.5 Characteristics of Interactions

We qualitatively coded interactions to determine whether GPT-4 clearly articulated one or more shortcomings in the sender's IPT rephrasing and whether the human sender addressed one or more shortcomings in response to the feedback.

6.5.1 Differences in LLM Interactivity from Human Interactivity. **GPT-4 interacted in ways that human receivers in Study 1 never did.** For example, GPT-4 would occasionally copyedit the entire sender rephrasing, which no human receiver either offered or attempted to do. There were also more subtle differences. GPT-4 would often give **many points** of feedback for the sender to clarify; in fact, GPT-4 gave *eight* points of feedback for one sender's IPT. This contrasts with Study 1, where a human would almost always request a *single* part of the IPT be clarified. Further, the **rate of interaction** was much higher in Study 2. The LLM gave feedback for 81.1% of the IPTs (146/180), whereas human receivers gave feedback for 26.6% of IPTs (68/256) in Study 1. When GPT-4 gave feedback, it gave an average of 2.2 distinct points of feedback. In contrast, human receivers were much more likely to pose a single request or single question. However, not all of GPT-4's feedback was sensible or relevant. For example, GPT-4 once asked, "Does the client need to pay for the free smoothie?" On average, GPT-4 provided 1.6 points of feedback per IPT that articulated a shortcoming of the sender's rephrasing. Thus, human senders sometimes needed to sift through GPT-4's feedback to determine what was relevant.

6.5.2 Differences in Interactivity Based on Programming Experience. GPT-4 seemed to interact differently with programmers and non-programmers. In particular, **GPT-4 gave feedback less frequently to programmers** (no feedback: 25.0%, 15/60 IPTs) than non-programmers (no feedback: 15.8%, 19/120 IPTs). Furthermore, the LLM gave programmers an average of 1.8 points of feedback and non-programmers an average of 2.4 points of feedback per IPT.

Programmers responded to at least one point of feedback 97.8% of the time (44/45). For those 45 IPTs, the senders on average responded to 1.8 points of feedback. In contrast, non-programmers responded to feedback 70.3% of the time (71/101). For those 101 IPTs, they responded on average to 1.2 points of feedback. We also saw differences in the degree to which programmers and non-programmers

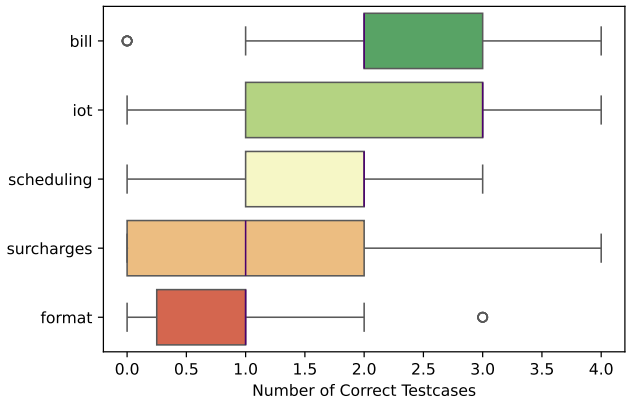

Figure 13: How many of the 4 test cases GPT-4 got right for each Study 2 IPT when *directly answering*.

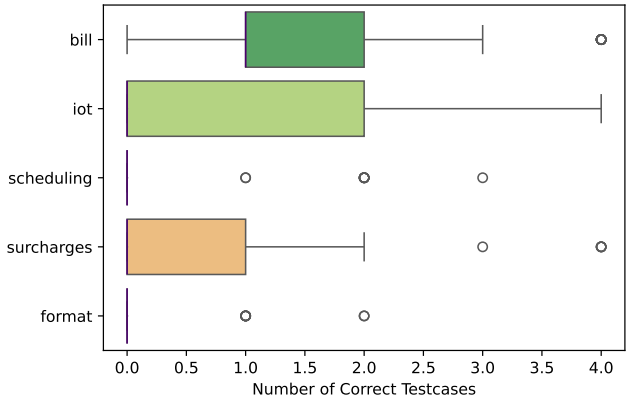

Figure 14: How many of the 4 test cases GPT-4 got right for each Study 2 IPT when *writing Python code*.

materially changed their descriptions. **Programmers were more likely to materially clarify their IPTs when responding** as they did so 81.4% of the time (35/44), while non-programmers materially clarified their IPTs 78.9% of the time (56/71). **Programmers also tended to materially clarify their IPTs to a greater degree** than non-programmers when responding, addressing on average 1.4 points of feedback, while non-programmers responded on average to 1.1 points of feedback.

6.5.3 LLMs Directly Answering vs. Producing Code. Finally, we revisit GPT-4's performance when directly answering test cases versus producing Python code to answer test cases. Recall that in Study 1 (Table 4 in Section 4.7), GPT-4 answered 59.8% of test cases correctly when directly answering, versus 45.6% of test cases by producing Python code. While GPT-4 essentially answered one-third more test cases correctly when directly answering in Study 1, for Study 2 we observed a far more pronounced difference. As a comparison of Figure 13 and Figure 14 shows, GPT-4 answered more than twice as many test cases correctly when directly answering (39.7% of test cases) than when producing Python code (17.6%).

6.6 Study 2: Should LLMs Have Failed?

Similar to Section 4.8, we examine 100 randomly selected IPT descriptions from Study 2 to determine whether cases where GPT-4 answered incorrectly might be fixed by improvements to LLMs, or whether the human senders' descriptions are insufficient. We observe a greater spread in the quality of IPTs in Study 2 than in Study 1. Namely, only 22/100 descriptions (vs. 45/100 in Study 1) contained **all information needed** to answer regular and edge cases, whereas 36/100 descriptions (vs. 33/100 in Study 1) contained **none of the information needed**. When the descriptions contained all information needed, senders correctly answered 3.3/4 test cases on average, whereas GPT-4 on average answered only 1.6/4 when directly answering and 0.7/4 when producing code. More senders forgot to specify the goal of the IPT in Study 2 than in Study 1, perhaps owing to the picture format. An example of an IPT rephrasing with an unclear goal was "Members pay a $1 surcharge on the items that are not $5 and $1 surcharge if they don't bring their own bag." While it can potentially be inferred that the IPT concerns calculating surcharges, this is never explicitly stated. The goal was unclear for 23/100 descriptions in Study 2, versus 2/100 in Study 1. Further, it appears that GPT-4 was able to correctly guess at the goal in these cases: the sender answered 1.3/4 test cases correctly on average, while GPT-4 answered on average 1.5/4 when directly answering and 0.9/4 when producing code.

Much like in Study 1, senders often ended up communicating different IPTs. This happened in 41/100 descriptions in our random sample for Study 2, as well as 31/100 in Study 1. Similarly, senders sometimes had missing definitions, which happened in 25/100 descriptions for Study 2 and 26/100 in Study 1. We again find that GPT-4 seemed to guess correctly in these cases. On average, senders got 1.4/4 test cases correct when there were missing definitions, but GPT-4 got 1.9/4 correct when directly answering and 1.1/4 when producing code. While extraneous information was less common in Study 2 (9/100 descriptions, vs. 18/100 in Study 1), senders who included extraneous information tended to do poorly on test cases, answering 1.3/4 correctly on average. Much like Study 1, senders with clearly formatted descriptions (13/100 IPTs) tended to demonstrate better understanding, answering 2.9/4 test cases correct on average. Curiously, GPT-4 seemed to do an abysmal job at producing code when the description was clearly formatted, answering on average 0.2/4 test cases correctly (1.4 when directly answering). However, senders who wrote their description in a step-by-step format (16/100 in Study 2) and who wrote their IPTs such that it was possible to answer all edge cases (22/100 in Study 2 vs. 48/100 in Study 1) tended to demonstrate clear understanding, answering on average 3.2/4 and 3.3/4 of test cases correctly, respectively. LLM performance for those descriptions remained low, however, with averages $\leq$ 1.6/4 regardless of whether they were producing code.

7 Discussion

We find evidence that a human sender's understanding of an IPT is a necessary prerequisite for either human or LLM receivers to understand the IPT. While this has been the topic of recent theorization by Sarkar, we provide empirical evidence to its necessity for end-user programming with LLMs [51]. In both Study 1 and Study 2, the number of test cases answered correctly by the sender was a

significant predictor of the number of test cases the receiver would get right regardless of whether the receiver was a human or LLM, as well as regardless of whether the LLM was writing code or directly answering. This suggests that clear IPT communication may be a necessary condition for enabling natural language programming. We expect that this result will hold regardless of how LLM technology improves as humans generally struggle to completely describe IPTs (Sections 4.8, 6.6). However, we also speculate that as LLMs improve, they may become more adept at guessing — we speculate this may be particularly relevant for failure modes relating to extraneous information and forgetting to specify the goal of the IPT. We also note that while it is well-known that examples are beneficial *when prompting LLMs* [5, 64], carefully selecting examples may be an under-explored option for clarifying the understanding and intent of *those prompting* LLMs. That is, examples could be useful for clarifying a sender's *intent*.

Critically, we do *not* find evidence that only individuals with programming experience are capable of understanding or communicating IPTs, although we do find that programmers consistently do a bit better than non-programmers at answering test cases and communicating IPTs. We find that programmers are more likely to include examples, clearly format examples, and include a larger number of examples. Further, programmers are less likely to receive feedback from either a human or LLM. However, we find that GPT-4 gives worse feedback to individuals with programming experience (i.e., the feedback tended to be irrelevant and/or incorrect at a higher rate than for non-programmers). Further, the LLM in our study—which was admittedly not asked to provide only a single point of feedback—often gave many points of feedback, up to seven or eight in some cases. These issues highlight a complex design matrix. Namely, we observe that feedback utility and quantity *decreased* with user expertise; inexperienced users may get overwhelmed by a large quantity of useful feedback, while experienced users may only be presented with a small quantity of ineffective feedback. We speculate that problems related to vast quantities of feedback may be mitigated in the future as alignment research continues. However, we also speculate that the gap between feedback *utility* for programmers and non-programmers will remain as there is likely to be extreme variance in what feedback is useful for non-programmers due to their absence of formal training.

We also note that the way in which LLMs enable end-user programming may need to change adaptively based on the nature of the IPT. We found in Study 1 that certain simple IPTs were trivial to express. However, Study 2's IPTs were inherently more complex, and GPT-4 struggled to produce functional code for many of the IPTs. We recommend that designers gauge whether code should be produced, or direct responses would be more appropriate based on task complexity as well as other artifacts that may suggest a lack of understanding on the part of the end user, such as incoherence in the input prompt. In this vein, we note from Study 2 that the quantity of feedback elements correlated significantly with if the model was likely to answer test cases correctly, either by producing code or by direct answering. We speculate that designers could simply prompt a model for feedback and record the number of points of feedback the model gives (without actually showing that feedback to the end user) in order to potentially gain an indication of whether the LLM is likely to respond correctly or not. A designer could then take actions like preemptively showing or hiding widgets based on the consequences of an incorrect response. We generally expect this result to hold at a high level in the future, although it may be the case that an "ideal" UI changes frequently due to specific idiosyncrasies of LLM outputs, such as consistently outputting too much text. This is further supported by our analysis in Sections 4.8 and 6.6, which shows that humans often forget to provide necessary definitions when describing IPTs and in those cases, the LLMs tend to struggle to produce correct code (which is to be expected given the pedantic nature of coding).

Related to the previous points is our observation that GPT-4 was more successful in Study 1 (Section 4.7) and far more successful in Study 2 (Section 6.5.3) when directly answering test cases rather than producing Python code to answer those test cases. This difference is particularly notable because, in the broader context of end-user programming, code could conceivably serve as a useful intermediate representation for the process learned by the LLM. For instance, the user could communicate their intention in natural language, have the LLM produce code as an internal representation, and then perform future computations using that cached code. Doing so has the advantage that the outputs created using the code representation would not confabulate (hallucinate) and also has the potential to provide greater transparency to the user about the representation learned, such as through the iterative generation of appropriate examples. As Study 2's IPTs were more complex and much less likely to have been previously seen in the LLM training process, particularly as half of the Study 1 IPTs were included in the HumanEval [7] benchmark, we believe that Study 2 is a more realistic view of future uses of LLMs for end-user programming.

8 Conclusion

In this paper, we examined the communication of information processing tasks to other humans, as well as to code-generating LLMs. We found that while programming experience played a beneficial role in communicating these tasks to other humans or LLMs, the effect was often complemented by, and sometimes secondary to, the effects of examples. We also found that programmers were more likely to clearly format their examples, were more likely to send examples, and were more likely to include a greater number of examples. We surprisingly found that allowing for interactivity did not play a significant role. Often, the presence of an interaction served as a (statistically significant) indicator that the sender's description was unclear. We found differences in how human receivers and LLMs provide feedback to senders. Notably, we found that LLM feedback tended to be irrelevant more frequently for programmers, and that the LLM tended to provide feedback more frequently for non-programmers. We posit that this may reveal a usability issue wherein novice users are more likely to be flooded with too much feedback while experienced users may receive a more manageable quantity of incorrect or irrelevant feedback. Finally, we speculate that because the effect of programming experience was often complemented by, or secondary to, the effect of examples, this programming experience "gap" might be bridged by nudging non-programmers towards programmers' beneficial behaviors through carefully designed user interfaces.

References

[1] Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, and Charles Sutton. 2021. Program Synthesis with Large Language Models. arXiv:2108.07732.

[2] Tewodros W. Ayalew, Jennifer Wang, Michael L. Littman, Blase Ur, and Sarah Sebo. 2025. Enabling End Users to Program Robots Using Reinforcement Learning. In *Proceedings of the 2025 ACM/IEEE International Conference on Human-Robot Interaction*.

[3] Yasharth Bajpai, Bhavya Chopra, Param Biyani, Cagri Aslan, Dustin Coleman, Sumit Gulwani, Chris Parnin, Arjun Radhakrishna, and Gustavo Soares. 2024. Let's Fix this Together: Conversational Debugging with GitHub Copilot. In *Proceedings of the 2024 IEEE Symposium on Visual Languages and Human-Centric Computing*.

[4] Barbara Rita Barricelli, Fabio Cassano, Daniela Fogli, and Antonio Piccinno. 2019. End-User Development, End-User Programming and End-User Software Engineering: A Systematic Mapping Study. *Journal of Systems and Software* 149 (2019), 101–137.

[5] 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. 2020. Language Models are Few-Shot Learners. arXiv:2005.14165.

[6] Sarina Canelake. 2011. A Gentle Introduction To Programming Using Python. https://ocw.mit.edu/courses/6-189-a-gentle-introduction-to-programming-using-python-january-iap-2011/

[7] Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian, Clemens Winter, Philippe Tillet, Felipe Petroski Such, Dave Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William Hebgen Guss, Alex Nichol, Alex Paino, Nikolas Tezak, Jie Tang, Igor Babuschkin, Suchir Balaji, Shantanu Jain, William Saunders, Christopher Hesse, Andrew N. Carr, Jan Leike, Josh Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. 2021. Evaluating Large Language Models Trained on Code. arXiv:2107.03374.

[8] Anastasia Danilova, Alena Naiakshina, Stefan Horstmann, and Matthew Smith. 2021. Do You Really Code? Designing and Evaluating Screening Questions for Online Surveys with Programmers. In *Proceedings of the 2021 IEEE/ACM 43rd International Conference on Software Engineering*.

[9] Ian Drosos, Advait Sarkar, Xiaotong Xu, Carina Negreanu, Sean Rintel, and Lev Tankelevitch. 2024. "It's Like a Rubber Duck That Talks Back": Understanding Generative AI-Assisted Data Analysis Workflows through a Participatory Prompting Study. In *Proceedings of the 3rd Annual Meeting of the Symposium on Human-Computer Interaction for Work*.

[10] Jean-Baptiste Döderlein, Mathieu Acher, Djamel Eddine Khelladi, and Benoit Combemale. 2023. Piloting Copilot and Codex: Hot Temperature, Cold Prompts, or Black Magic? arXiv:2210.14699.

[11] Kasra Ferdowsi, Jack Williams, Ian Drosos, Andy Gordon, Carina Negreanu, Nadia Polikarpova, Advait Sarkar, and Ben Zorn. 2023. ColDeco: An End User Spreadsheet Inspection Tool for AI-Generated Code. In *Proceedings of the IEEE Symposium on Visual Languages and Human-Centric Computing*.

[12] James Finnie-Ansley, Paul Denny, Brett A. Becker, Andrew Luxton-Reilly, and James Prather. 2022. The Robots Are Coming: Exploring the Implications of OpenAI Codex on Introductory Programming. In *Proceedings of the 24th Australasian Computing Education Conference*.

[13] James Finnie-Ansley, Paul Denny, Andrew Luxton-Reilly, Eddie Antonio Santos, James Prather, and Brett A. Becker. 2023. My AI Wants to Know if This Will Be on the Exam: Testing OpenAI's Codex on CS2 Programming Exercises. In *Proceedings of the 25th Australasian Computing Education Conference*.

[14] Neil Fraser. 2015. Ten Things We've Learned From Blockly. In *Proceedings of the 2015 IEEE Blocks and Beyond Workshop*.

[15] Gemini Team Google. 2024. Gemini: A Family of Highly Capable Multimodal Models. arXiv:2312.11805.

[16] Dan Hendrycks, Steven Basart, Saurav Kadavath, Mantas Mazeika, Akul Arora, Ethan Guo, Collin Burns, Samir Puranik, Horace He, Dawn Song, and Jacob Steinhardt. 2021. Measuring Coding Challenge Competence With APPS. arXiv:2105.09938.

[17] Justin Huang and Maya Cakmak. 2015. Supporting Mental Model Accuracy in Trigger-Action Programming. In *Proceedings of the 2015 ACM International Joint Conference on Pervasive and Ubiquitous Computing*.

[18] Dhanya Jayagopal, Justin Lubin, and Sarah E. Chasins. 2022. Exploring the Learnability of Program Synthesizers by Novice Programmers. In *Proceedings of the 35th Annual ACM Symposium on User Interface Software and Technology*.

[19] Eirini Kalliamvakou. 2022. Research: Quantifying GitHub Copilot's Impact on Developer Productivity and Happiness. https://github.blog/2022-09-07-research-quantifying-github-copilots-impact-on-developer-productivity-and-happiness/

[20] Ulas Berk Karli, Juo-Tung Chen, Victor Nikhil Antony, and Chien-Ming Huang. 2024. Alchemist: LLM-Aided End-User Development of Robot Applications. In *Proceedings of the 2024 ACM/IEEE International Conference on Human-Robot Interaction*.

[21] Mohammad Amin Kuhail, Shahbano Farooq, Rawad Hammad, and Mohammed Bahja. 2021. Characterizing Visual Programming Approaches For End-User Developers: A Systematic Review. *IEEE Access* 9 (2021), 14181–14202.

[22] 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. 2024. Evaluating Human-Language Model Interaction. arXiv:2212.09746.

[23] LeetCode. 2023. LeetCode: A New Way to Learn. https://leetcode.com/

[24] Nicola Leonardi, Marco Manca, Fabio Paternò, and Carmen Santoro. 2019. Trigger-Action Programming For Personalising Humanoid Robot Behaviour. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*.

[25] Henry Lieberman, Fabio Paternò, Markus Klann, and Volker Wulf. 2006. End-User Development: An Emerging Paradigm. In *End User Development*. Springer.

[26] Greg Little, Tessa A. Lau, Allen Cypher, James Lin, Eben M. Haber, and Eser Kandogan. 2007. Koala: Capture, Share, Automate, Personalize Business Processes on the Web. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*.

[27] Michael Xieyang Liu, Advait Sarkar, Carina Negreanu, Benjamin Zorn, Jack Williams, Neil Toronto, and Andrew D. Gordon. 2023. "What It Wants Me To Say": Bridging the Abstraction Gap Between End-User Programmers and Code-Generating Large Language Models. In *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems*.

[28] Andrew M. McNutt, Chenglong Wang, Robert A. Deline, and Steven M. Drucker. 2023. On the Design of AI-Powered Code Assistants For Notebooks. In *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems*.

[29] André N. Meyer, Earl T. Barr, Christian Bird, and Thomas Zimmermann. 2021. Today Was a Good Day: The Daily Life of Software Developers. *IEEE Transactions on Software Engineering* 47, 5 (2021), 863–880.

[30] Xianghang Mi, Feng Qian, Ying Zhang, and XiaoFeng Wang. 2017. An Empirical Characterization of IFTTT: Ecosystem, Usage, and Performance. In *Proceedings of the 2017 Internet Measurement Conference*.

[31] Rob Miller, Victoria H. Chou, Michael S. Bernstein, Greg Little, Max Van Kleek, David R. Karger, and mc schraefel. 2008. Inky: A Sloppy Command Line For the Web With Rich Visual Feedback. In *Proceedings of the ACM Symposium on User Interface Software and Technology*.

[32] Arghavan Moradi Dakhel, Vahid Majdinasab, Amin Nikanjam, Foutse Khomh, Michel C. Desmarais, and Zhen Ming (Jack) Jiang. 2023. GitHub Copilot AI Pair Programmer: Asset or Liability? *Journal of Systems and Software* 203 (2023).

[33] Brad A. Myers, Amy J. Ko, and Margaret M. Burnett. 2006. Invited Research Overview: End-User Programming. In *Extended Abstracts of the 2006 CHI Conference on Human Factors in Computing Systems*.

[34] Bonnie A. Nardi. 1993. *A Small Matter of Programming: Perspectives on End User Computing*. MIT Press.

[35] Nhan Nguyen and Sarah Nadi. 2022. An Empirical Evaluation of GitHub Copilot's Code Suggestions. In *Proceedings of the 19th International Conference on Mining Software Repositories*.

[36] OpenAI. 2022. Introducing ChatGPT. https://openai.com/blog/chatgpt

[37] OpenAI. 2024. GPT-4 Technical Report. arXiv:2303.08774

[38] Hammond Pearce, Baleegh Ahmad, Benjamin Tan, Brendan Dolan-Gavitt, and Ramesh Karri. 2021. Asleep at the Keyboard? Assessing the Security of GitHub Copilot's Code Contributions. arXiv:2108.09293

[39] Neil Perry, Megha Srivastava, Deepak Kumar, and Dan Boneh. 2023. Do Users Write More Insecure Code With AI Assistants?. In *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security*.

[40] Madison Pickering, Helena Williams, Alison Gan, Weijia He, Hyojae Park, Francisco Piedrahita Velez, Michael L. Littman, and Blase Ur. 2025. Data Release. https://github.com/UChicagoSUPERgroup/chi25

[41] Prolific. 2025. Prolific: A Higher Standard of Online Research. https://www.prolific.co/

[42] Alec Radford, Jeff Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. 2019. Language Models are Unsupervised Multitask Learners. OpenAI Blog.

[43] Amir Rahmati, Earlence Fernandes, Jaeyeon Jung, and Atul Prakash. 2017. IFTTT vs. Zapier: A Comparative Study of Trigger-Action Programming Frameworks. arXiv:1709.02788.

[44] Mitchel Resnick, John Maloney, Andrés Monroy-Hernández, Natalie Rusk, Evelyn Eastmond, Karen Brennan, Amon Millner, Eric Rosenbaum, Jay Silver, Brian

Silverman, and Yasmin Kafai. 2009. Scratch: Programming For All. *Commun. ACM* 52, 11 (2009), 60–67.

[45] Kevin Roose. 2023. How Schools Can Survive (and Maybe Even Thrive) With A.I. This Fall. New York Times. https://www.nytimes.com/2023/08/24/technology/how-schools-can-survive-and-maybe-even-thrive-with-ai-this-fall.html

[46] Steven I. Ross, Fernando Martinez, Stephanie Houde, Michael Muller, and Justin D. Weisz. 2023. The Programmer's Assistant: Conversational Interaction with a Large Language Model for Software Development. In *Proceedings of the 28th International Conference on Intelligent User Interfaces*.

[47] Baptiste Roziere, Marie-Anne Lachaux, Lowik Chanussot, and Guillaume Lample. 2020. Unsupervised Translation of Programming Languages. In *Proceedings of the 34th International Conference on Neural Information Processing Systems*.

[48] 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. 2024. Code Llama: Open Foundation Models for Code. arXiv:2308.12950

[49] Gustavo Sandoval, Hammond Pearce, Teo Nys, Ramesh Karri, Siddharth Garg, and Brendan Dolan-Gavitt. 2023. Lost at C: A User Study on the Security Implications of Large Language Model Code Assistants. In *Proceedings of the 32nd USENIX Security Symposium*.

[50] Advait Sarkar. 2023. Will Code Remain a Relevant User Interface for End-User Programming with Generative AI Models?. In *Proceedings of the 2023 ACM SIGPLAN International Symposium on New Ideas, New Paradigms, and Reflections on Programming and Software*.

[51] Advait Sarkar. 2024. Intention Is All You Need. arXiv:2410.18851

[52] Advait Sarkar, Andrew D. Gordon, Carina Negreanu, Christian Poelitz, Sruti Srinivasa Ragavan, and Ben Zorn. 2022. What Is It Like to Program With Artificial Intelligence? arXiv:2208.06213

[53] Sami Sarsa, Paul Denny, Arto Hellas, and Juho Leinonen. 2022. Automatic Generation of Programming Exercises and Code Explanations Using Large Language Models. In *Proceedings of the 2022 ACM Conference on International Computing Education Research*.

[54] Aarohi Srivastava et al. 2022. Beyond the Imitation Game: Quantifying and Extrapolating the Capabilities of Language Models. arXiv:2206.04615.

[55] Florian Tambon, Arghavan Moradi Dakhel, Amin Nikanjam, Foutse Khomh, Michel C. Desmarais, and Giuliano Antoniol. 2024. Bugs in Large Language Models Generated Code: An Empirical Study. arXiv:2403.08937

[56] Lev Tankelevitch, Viktor Kewenig, Auste Simkute, Ava Elizabeth Scott, Advait Sarkar, Abigail Sellen, and Sean Rintel. 2024. The Metacognitive Demands and Opportunities of Generative AI. In *Proceedings of the 2024 CHI Conference on Human Factors in Computing Systems*.

[57] 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. 2023. LLaMA: Open and Efficient Foundation Language Models. arXiv:2302.13971

[58] Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, Dan Bikel, Lukas Blecher, Cristian Canton Ferrer, Moya Chen, Guillem Cucurull, David Esiobu, Jude Fernandes, Jeremy Fu, Wenyin Fu, Brian Fuller, Cynthia Gao, Vedanuj Goswami, Naman Goyal, Anthony Hartshorn, Saghar Hosseini, Rui Hou, Hakan Inan, Marcin Kardas, Viktor Kerkez, Madian Khabsa, Isabel Kloumann, Artem Korenev, Punit Singh Koura, Marie-Anne Lachaux, Thibaut Lavril, Jenya Lee, Diana Liskovich, Yinghai Lu, Yuning Mao, Xavier Martinet, Todor Mihaylov, Pushkar Mishra, Igor Molybog, Yixin Nie, Andrew Poulton, Jeremy Reizenstein, Rashi Rungta, Kalyan Saladi, Alan Schelten, Ruan Silva, Eric Michael Smith, Ranjan Subramanian, Xiaoqing Ellen Tan, Binh Tang, Ross Taylor, Adina Williams, Jian Xiang Kuan, Puxin Xu, Zheng Yan, Iliyan Zarov, Yuchen Zhang, Angela Fan, Melanie Kambadur, Sharan Narang, Aurelien Rodriguez, Robert Stojnic, Sergey Edunov, and Thomas Scialom. 2023. Llama 2: Open Foundation and Fine-Tuned Chat Models. arXiv:2307.09288.

[59] Blase Ur, Elyse McManus, Melwyn Pak Yong Ho, and Michael L. Littman. 2014. Practical Trigger-Action Programming in the Smart Home. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*.

[60] Blase Ur, Melwyn Pak Yong Ho, Stephen Brawner, Jiyun Lee, Sarah Mennicken, Noah Picard, Diane Schulze, and Michael L. Littman. 2016. 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*.

[61] Priyan Vaithilingam, Tianyi Zhang, and Elena L. Glassman. 2022. Expectation vs. Experience: Evaluating the Usability of Code Generation Tools Powered by Large Language Models. In *Extended Abstracts of the 2022 CHI Conference on Human Factors in Computing Systems*.

[62] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention Is All You Need. arXiv:1706.03762.

[63] Jeffrey Wong and Jason I. Hong. 2007. Making Mashups With Marmite: Towards End-User Programming For the Web. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*.

[64] J.D. Zamfirescu-Pereira, Richmond Y. Wong, Bjoern Hartmann, and Qian Yang. 2023. Why Johnny Can't Prompt: How Non-AI Experts Try (and Fail) to Design LLM Prompts. In *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems*.

[65] Lefan Zhang, Weijia He, Jesse Martinez, Noah Brackenbury, Shan Lu, and Blase Ur. 2019. AutoTap: Synthesizing and Repairing Trigger-Action Programs Using LTL Properties. In *Proceedings of the 2019 IEEE/ACM 41st International Conference on Software Engineering*.

[66] Lefan Zhang, Weijia He, Olivia Morkved, Valerie Zhao, Michael L. Littman, Shan Lu, and Blase Ur. 2020. Trace2tap: Synthesizing Trigger-Action Programs From Traces of Behavior. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies* 4, 3 (2020).

[67] Valerie Zhao, Lefan Zhang, Bo Wang, Michael L Littman, Shan Lu, and Blase Ur. 2021. Understanding Trigger-Action Programs Through Novel Visualizations of Program Differences. In *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems*.

A Additional Details About Selecting IPTs For Study 1

We aimed to ensure our programming tasks were representative of problems that programmers would likely 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 (APPS) or a high tendency to invoke mathematical or algorithmic definitions (MBPP and BigBench). The set of n=150 questions was then randomly ordered. Two researchers inductively developed the codebook in Table 14 to describe the types of problems, and qualitatively coded the questions. The researchers then met to resolve differences.

Table 14: 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.

B Full Descriptions of IPTs Given to Senders in Study 1

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 [7].

(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 HumanEval 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 door frame. 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', 'ttttttttt'], 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]]**

C Codebook Developed For Qualitative Analysis in Study 1

Qualitative Code	Description
All Information	Contains all the information necessary to solve the test cases
Some Information	Missing some of the information necessary to solve the test cases
No Information	Contains none of the information necessary to solve the test cases; may simply repeat the context
Generalized All	The IPT rephrasing is fully generalizable; none of the context remains
Generalized Some	The IPT rephrasing has been somewhat generalized, but some context remains (e.g., given participant's score totals, find the max)
Generalized None	The IPT rephrasing has not been generalized; it is highly tied to its context
Correct Examples	The IPT rephrasing contains examples not corresponding to edge cases; these examples are correct
Incorrect Examples	The IPT rephrasing contains examples not corresponding to edge cases, but one or more is incorrect
Edge Example Correct	The IPT rephrasing contains examples corresponding to an edge case; these examples are correct
Edge Example Incorrect	The IPT rephrasing contains examples corresponding to an edge case, but one or more is incorrect
Contains Extra Information	The IPT rephrasing contains extraneous information
Concise Attempt	A noticeable attempt was made to make the IPT rephrasing more concise
Steps	The IPT rephrasing includes steps to solve the task
No Meaningful Information	The IPT rephrasing is devoid of any meaningful information
ChatGPT	The IPT rephrasing seems to have been written by ChatGPT or another AI tool
Exact Match	The participant essentially re-wrote the IPT rephrasing the researchers provided with the same examples (when applicable)

D Depictions of IPTs Given to Senders in Study 2

Figure 15: The `bill` IPT.

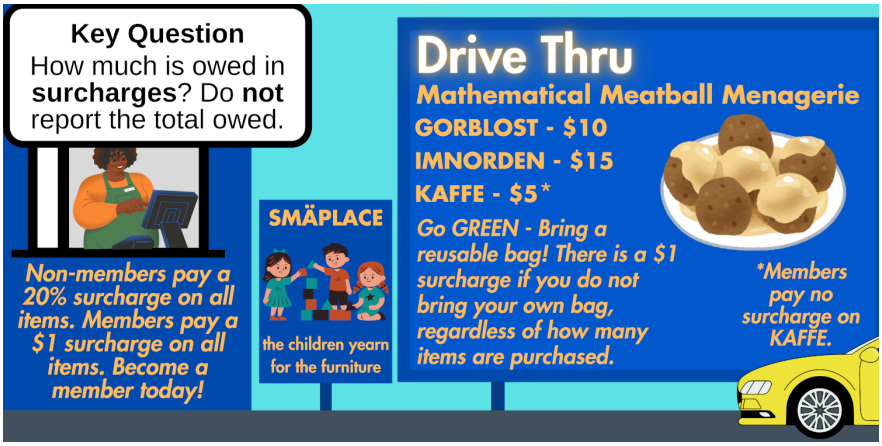

Figure 16: The `surcharges` IPT.

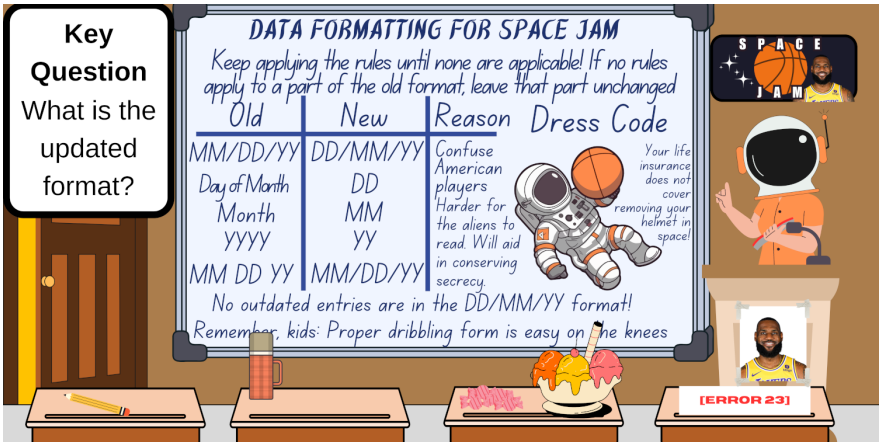

Figure 17: The `format` IPT.

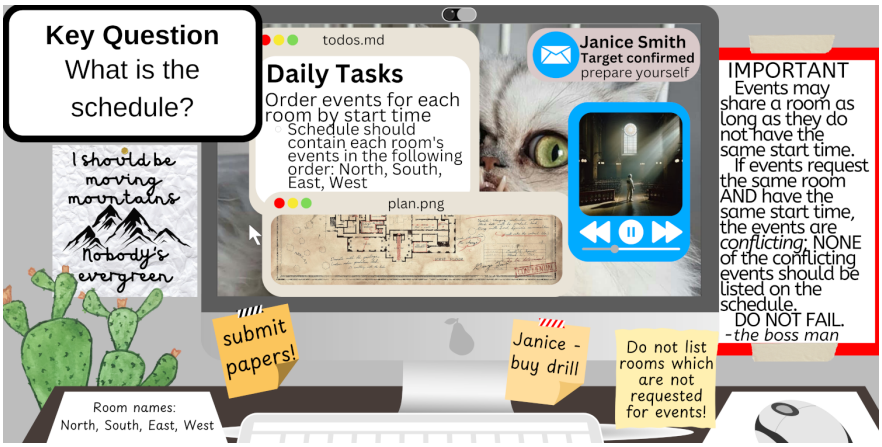

Figure 18: The `scheduling` IPT.

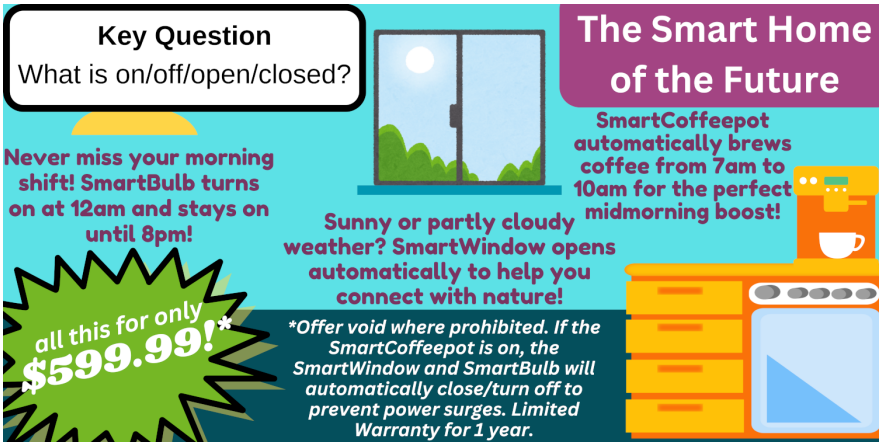

Figure 19: The `iot` IPT.

E Study 2 Participant Views

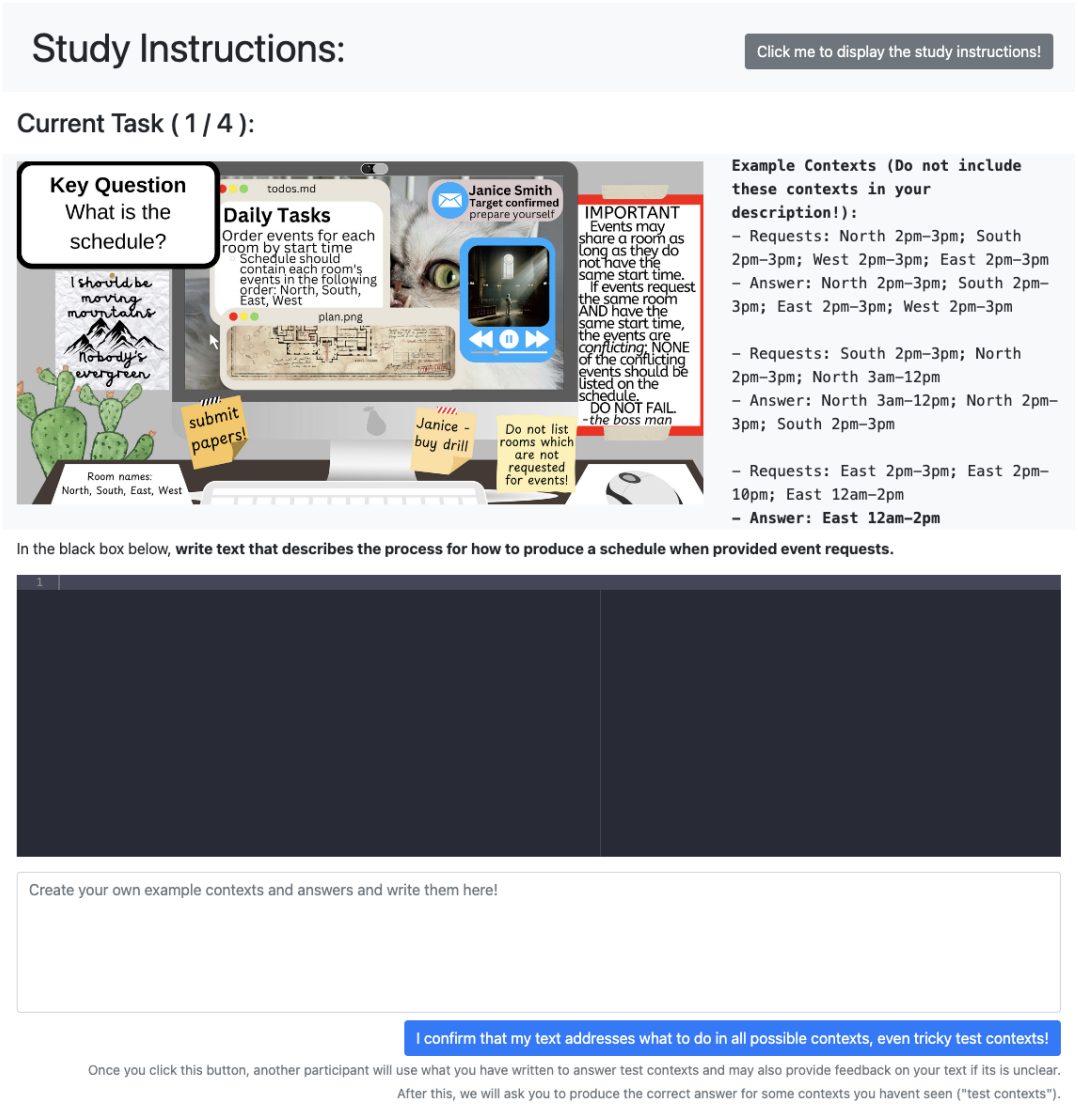

Figure 20: The sender view with instructions collapsed.

Figure 21: The sender view with instructions expanded.

F Additional Graphs of Demographic Correlations in Study 1

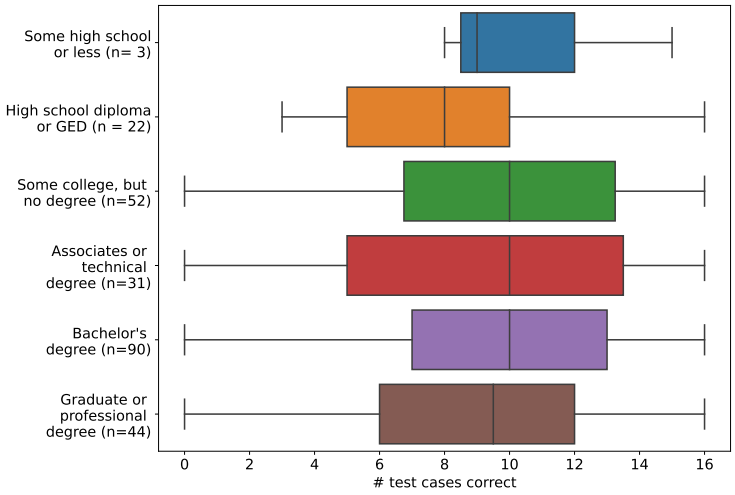

Figure 22: The number of test cases answered correctly split by participants' highest level of education.

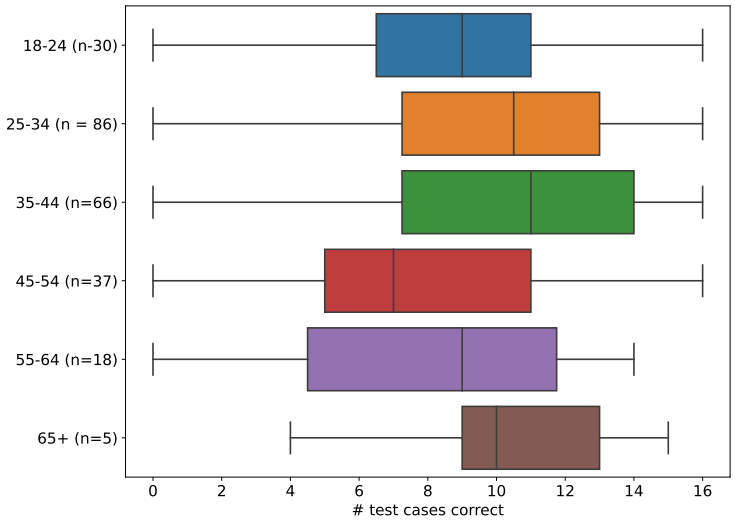

Figure 23: The number of test cases answered correctly split by participants' age range.

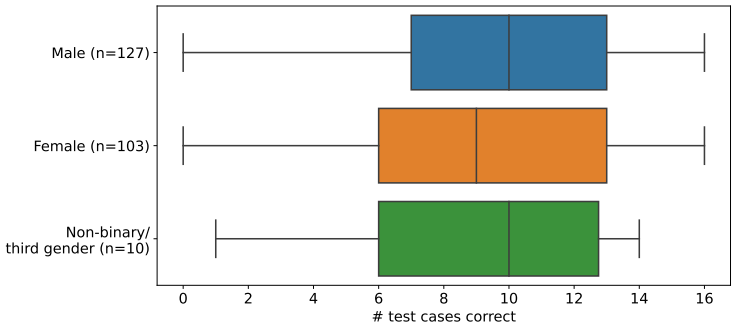

Figure 24: The number of test cases answered correctly split by participants' gender.

G Additional Regression Tables for LLMs in Study 1

Table 15: Linear regression with the dependent variable indicating how many of the four test cases *Code Llama 34B* answered correctly when *directly answering*. Adjusted R^2 = 0.299.

Independent Variable	Baseline	β	SE	t	p
(Intercept)	–	1.472	0.261	5.637	**<.001**
Condition: Examples	Non-examples	0.059	0.111	0.533	.594
Condition: Interactive	Non-interactive	0.030	0.107	0.281	.779
Sender: # Test Cases Correct	–	0.193	0.042	4.660	**<.001**
Sender is Programmer	Non-programmer	0.244	0.127	1.925	.055
Sender Age 18–24	25–34	-0.141	0.185	-0.759	.448
Sender Age 35–44	25–34	-0.044	0.129	-0.338	.735
Sender Age 45+	25–34	-0.413	0.153	-2.702	**.007**
Sender is Non-male	Male	0.044	0.113	0.393	.695
Sender Degree: None	Bachelor's	0.104	0.126	0.824	.410
Sender Degree: Associate's	Bachelor's	0.256	0.188	1.361	.174
Sender Degree: Graduate	Bachelor's	0.255	0.155	1.648	.100
IPT: `max`	`sum_pos`	0.008	0.243	0.032	.974
IPT: `num_even`	`sum_pos`	-1.242	0.261	-4.751	**<.001**
IPT: `reignfall`	`sum_pos`	0.390	0.273	1.428	.154
IPT: `palindrome`	`sum_pos`	0.126	0.251	0.505	.614
IPT: `find_sum`	`sum_pos`	-0.804	0.262	-3.071	**.002**
IPT: `rmv_dup`	`sum_pos`	-1.256	0.262	-4.796	**<.001**
IPT: `str_diff`	`sum_pos`	-0.759	0.252	-3.012	**.003**
IPT: `is_subseq`	`sum_pos`	0.050	0.255	0.197	.844
IPT: `exact_chg`	`sum_pos`	-0.084	0.247	-0.341	.733
IPT: `fizzbuzz`	`sum_pos`	-1.195	0.269	-4.450	**<.001**
IPT: `xyz`	`sum_pos`	-0.809	0.265	-3.050	**.002**

Table 16: Linear regression with the dependent variable indicating how many of the four test cases *Code Llama 34B* answered correctly when *producing Python code*. Adjusted R^2 = 0.446.

Independent Variable	Baseline	β	SE	t	p
(Intercept)	–	2.332	0.283	8.237	**<.001**
Condition: Examples	Non-examples	0.376	0.120	3.134	**.002**
Condition: Interactive	Non-interactive	-0.004	0.116	-0.033	.974
Sender: # Test Cases Correct	–	0.233	0.045	5.179	**<.001**
Sender is Programmer	Non-programmer	0.443	0.138	3.220	**.001**
Sender Age 18–24	25–34	0.003	0.201	0.013	.990
Sender Age 35–44	25–34	-0.091	0.140	-0.654	.513
Sender Age 45+	25–34	-0.459	0.166	-2.772	**.006**
Sender is Non-male	Male	-0.007	0.122	-0.059	.953
Sender Degree: None	Bachelor's	0.132	0.136	0.969	.333
Sender Degree: Associate's	Bachelor's	0.132	0.204	0.646	.519
Sender Degree: Graduate	Bachelor's	0.109	0.168	0.652	.515
IPT: `max`	`sum_pos`	-1.663	0.263	-6.323	**<.001**
IPT: `num_even`	`sum_pos`	-1.072	0.284	-3.783	**<.001**
IPT: `reignfall`	`sum_pos`	-1.307	0.296	-4.413	**<.001**
IPT: `palindrome`	`sum_pos`	0.311	0.272	1.145	.253
IPT: `find_sum`	`sum_pos`	-2.255	0.284	-7.946	**<.001**
IPT: `rmv_dup`	`sum_pos`	-2.888	0.284	-10.165	**<.001**
IPT: `str_diff`	`sum_pos`	-2.519	0.273	-9.216	**<.001**
IPT: `is_subseq`	`sum_pos`	-1.830	0.277	-6.617	**<.001**
IPT: `exact_chg`	`sum_pos`	-1.664	0.267	-6.221	**<.001**
IPT: `fizzbuzz`	`sum_pos`	-2.095	0.291	-7.194	**<.001**
IPT: `xyz`	`sum_pos`	-1.584	0.288	-5.506	**<.001**

Table 17: Linear regression with the dependent variable indicating how many of the four test cases *Gemini* answered correctly when *directly answering*. Adjusted R^2 = 0.388.

Independent Variable	Baseline	β	SE	t	p
(Intercept)	–	2.162	0.240	8.999	**<.001**
Condition: Examples	Non-examples	0.328	0.102	3.221	**.001**
Condition: Interactive	Non-interactive	0.126	0.098	1.279	.202
Sender: # Test Cases Correct	–	0.111	0.038	2.895	**.004**
Sender is Programmer	Non-programmer	0.078	0.117	0.667	.505
Sender Age 18–24	25–34	0.100	0.170	0.585	.559
Sender Age 35–44	25–34	-0.079	0.119	-0.665	.506
Sender Age 45+	25–34	-0.212	0.140	-1.510	.132
Sender is Non-male	Male	-0.123	0.104	-1.187	.236
Sender Degree: None	Bachelor's	0.124	0.116	1.068	.286
Sender Degree: Associate's	Bachelor's	-0.111	0.173	-0.640	.522
Sender Degree: Graduate	Bachelor's	0.087	0.142	0.613	.540
IPT: `max`	`sum_pos`	-0.254	0.223	-1.136	.256
IPT: `num_even`	`sum_pos`	-2.053	0.241	-8.534	**<.001**
IPT: `reignfall`	`sum_pos`	-2.114	0.251	-8.412	**<.001**
IPT: `palindrome`	`sum_pos`	-2.226	0.231	-9.656	**<.001**
IPT: `find_sum`	`sum_pos`	-0.755	0.241	-3.138	**.002**
IPT: `rmv_dup`	`sum_pos`	-1.800	0.241	-7.469	**<.001**
IPT: `str_diff`	`sum_pos`	-1.871	0.232	-8.069	**<.001**
IPT: `is_subseq`	`sum_pos`	-1.646	0.235	-7.017	**<.001**
IPT: `exact_chg`	`sum_pos`	-2.318	0.227	-10.216	**<.001**
IPT: `fizzbuzz`	`sum_pos`	-1.883	0.247	-7.622	**<.001**
IPT: `xyz`	`sum_pos`	-1.449	0.244	-5.939	**<.001**

Table 18: Linear regression with the dependent variable indicating how many of the four test cases *Gemini* answered correctly when *producing Python code*. Adjusted R^2 = 0.383.

Independent Variable	Baseline	β	SE	t	p
(Intercept)	–	2.398	0.308	7.776	**<.001**
Condition: Examples	Non-examples	0.317	0.131	2.426	**.016**
Condition: Interactive	Non-interactive	-0.168	0.126	-1.328	.185
Sender: # Test Cases Correct	–	0.228	0.049	4.657	**<.001**
Sender is Programmer	Non-programmer	0.395	0.150	2.637	**.009**
Sender Age 18–24	25–34	-0.159	0.219	-0.728	.467
Sender Age 35–44	25–34	-0.109	0.152	-0.719	.472
Sender Age 45+	25–34	-0.655	0.180	-3.631	**<.001**
Sender is Non-male	Male	-0.006	0.133	-0.043	.966
Sender Degree: None	Bachelor's	0.077	0.149	0.517	.606
Sender Degree: Associate's	Bachelor's	0.114	0.222	0.513	.608
Sender Degree: Graduate	Bachelor's	0.232	0.183	1.270	.205
IPT: `max`	`sum_pos`	-1.767	0.286	-6.167	**<.001**
IPT: `num_even`	`sum_pos`	-0.365	0.309	-1.181	.238
IPT: `reignfall`	`sum_pos`	-0.978	0.323	-3.030	**.003**
IPT: `palindrome`	`sum_pos`	0.419	0.296	1.414	.158
IPT: `find_sum`	`sum_pos`	-1.530	0.309	-4.951	**<.001**
IPT: `rmv_dup`	`sum_pos`	-2.675	0.309	-8.644	**<.001**
IPT: `str_diff`	`sum_pos`	-2.245	0.298	-7.539	**<.001**
IPT: `is_subseq`	`sum_pos`	-1.163	0.301	-3.861	**<.001**
IPT: `exact_chg`	`sum_pos`	-1.514	0.291	-5.196	**<.001**
IPT: `fizzbuzz`	`sum_pos`	-1.733	0.317	-5.464	**<.001**
IPT: `xyz`	`sum_pos`	-1.400	0.313	-4.468	**<.001**

H Qualitative Regression Tables for Study 1

H.1 Characteristics of Descriptions (All Conditions)

Table 19: Linear regression correlating our qualitative codes of the descriptions written by senders and how many of the four test cases *the (human) receiver* answered correctly in Study 1. Adjusted $R^2 = 0.218$.

Independent Variable	Baseline	β	SE	t	p
(Intercept)	–	2.695	0.255	10.567	**<.001**
Necessary Info: None	All	-1.928	0.229	-8.424	**<.001**
Necessary Info: Some	All	-1.074	0.145	-7.413	**<.001**
Generalized: Not At All	Completely	0.034	0.166	0.207	.836
Generalized: Partially	Completely	-0.419	0.193	-2.173	**.030**
Reused Our Description	Did Not	1.074	0.522	2.057	**.040**
Concise	Not Concise	0.039	0.213	0.184	.854
Lists Steps	None	0.123	0.349	0.353	.724
Contains Explanations	None	0.584	0.316	1.850	.065
Extraneous Info	None	-0.050	0.224	-0.222	.824

Table 20: Linear regression correlating our qualitative codes of the descriptions written by senders and how many of the four test cases *GPT-4* answered correctly in Study 1 when *directly answering*. Adjusted $R^2 = 0.256$.

Independent Variable	Baseline	β	SE	t	p
(Intercept)	–	3.066	0.237	12.961	**<.001**
Necessary Info: None	All	-2.019	0.212	-9.512	**<.001**
Necessary Info: Some	All	-1.043	0.134	-7.760	**<.001**
Generalized: Not At All	Completely	-0.101	0.154	-0.656	.512
Generalized: Partially	Completely	-0.284	0.179	-1.585	.114
Reused Our Description	Did Not	1.804	0.484	3.728	**<.001**
Concise	Not Concise	0.088	0.198	0.443	.658
Lists Steps	None	0.150	0.323	0.464	.643
Contains Explanations	None	0.298	0.293	1.017	.310
Extraneous Info	None	-0.124	0.208	-0.595	.552

Table 21: Linear regression correlating our qualitative codes of the descriptions written by senders and how many of the four test cases *GPT-4* answered correctly in Study 1 when *writing Python code*. Adjusted $R^2 = 0.323$.

Independent Variable	Baseline	β	SE	t	p
(Intercept)	–	3.339	0.262	12.742	**<.001**
Necessary Info: None	All	-2.245	0.235	-9.545	**<.001**
Necessary Info: Some	All	-1.575	0.149	-10.575	**<.001**
Generalized: Not At All	Completely	-0.634	0.171	-3.714	**<.001**
Generalized: Partially	Completely	-0.235	0.198	-1.186	.236
Reused Our Description	Did Not	1.914	0.536	3.570	**<.001**
Concise	Not Concise	-0.059	0.219	-0.271	.787
Lists Steps	None	0.037	0.358	0.104	.917
Contains Explanations	None	-0.447	0.324	-1.378	.169
Extraneous Info	None	0.195	0.231	0.844	.399

H.2 Characteristics of Examples (Condition: Examples)

For the following tables, we mark a pair as having (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.

Table 22: Linear regression correlating our qualitative codes of the descriptions written by senders in the *examples* condition and how many of the four test cases *the (human) receiver* answered correctly in Study 1. Adjusted $R^2 = 0.059$.

Independent Variable	Baseline	β	SE	t	p
(Intercept)	–	2.384	0.390	6.117	**<.001**
# of Examples	–	0.162	0.104	1.557	.121
Did Not Send Examples	Did Send	-0.373	0.422	-0.884	.378
Reused Any of Our Examples	Did Not	-0.750	0.376	-1.998	**.047**
Included Context	Did Not	1.005	0.653	1.538	.126
Explained Examples	Did Not	0.231	0.378	0.613	.541
Properly Formatted	Not	-0.111	0.373	-0.298	.766

Table 23: Linear regression correlating our qualitative codes of the descriptions written by senders in the *examples* condition and how many of the four test cases *GPT-4* answered correctly in Study 1 when *directly answering*. Adjusted $R^2 = 0.089$.

Independent Variable	Baseline	β	SE	t	p
(Intercept)	–	1.302	0.351	3.706	**<.001**
# of Examples	–	0.314	0.094	3.341	**.001**
Did Not Send Examples	Did Send	1.237	0.380	3.255	**.001**
Reused Any of Our Examples	Did Not	0.358	0.339	1.057	.292
Included Context	Did Not	0.373	0.589	0.633	.528
Explained Examples	Did Not	0.952	0.341	2.795	**.006**
Properly Formatted	Not	0.483	0.336	1.437	.152

Table 24: Linear regression correlating our qualitative codes of the descriptions written by senders in the *examples* condition and how many of the four test cases *GPT-4* answered correctly in Study 1 when *writing Python code*. Adjusted $R^2 = 0.096$.

Independent Variable	Baseline	β	SE	t	p
(Intercept)	–	1.309	0.410	3.195	**.002**
# of Examples	–	0.305	0.110	2.785	**.006**
Did Not Send Examples	Did Send	0.837	0.443	1.890	.060
Reused Any of Our Examples	Did Not	0.012	0.395	0.030	.976
Included Context	Did Not	-1.043	0.687	-1.519	.131
Explained Examples	Did Not	0.418	0.397	1.052	.294
Properly Formatted	Not	0.738	0.392	1.884	.061

H.3 Characteristics of Interactions (Condition: Interactive)

Table 25: Linear regression correlating our qualitative codes of the descriptions written by senders in the *interactive* condition and how many of the four test cases *the (human) receiver* answered correctly in Study 1. Adjusted $R^2 = 0.017$.

Independent Variable	Baseline	β	SE	t	p
(Intercept)	–	2.210	0.152	14.564	**<.001**
Interacted: No	Yes	0.473	0.231	2.045	**.042**
# Clarification Requests	–	-0.090	1.018	-0.089	.930
# Clarification Responses	–	-0.710	1.108	-0.641	.522
Sender Edited Description	Did Not	-0.650	0.934	-0.696	.487

Table 26: Linear regression correlating our qualitative codes of the descriptions written by senders in the *interactive* condition and how many of the four test cases *GPT-4* answered correctly in Study 1 when *directly answering*. Adjusted $R^2 = 0.022$.

Independent Variable	Baseline	β	SE	t	p
(Intercept)	–	2.492	0.139	17.980	**<.001**
Interacted: No	Yes	0.458	0.211	2.166	**.032**
# Clarification Requests	–	1.361	0.930	1.464	.145
# Clarification Responses	–	-0.992	1.012	-0.980	.328
Sender Edited Description	Did Not	-1.066	0.853	-1.250	.213

Table 27: Linear regression correlating our qualitative codes of the descriptions written by senders in the *interactive* condition and how many of the four test cases *GPT-4* answered correctly in Study 1 when *writing Python code*. Adjusted $R^2 = -0.008$.

Independent Variable	Baseline	β	SE	t	p
(Intercept)	–	2.287	0.165	13.891	**<.001**
Interacted: No	Yes	0.333	0.251	1.328	.186
# Clarification Requests	–	-0.123	1.104	-0.111	.911
# Clarification Responses	–	-0.787	1.202	-0.655	.513
Sender Edited Description	Did Not	0.295	1.013	0.291	.771

I Survey Instruments

I.1 Study 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.

I.2 Study 1 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.

I.3 Study 2 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 computational tasks.

You will now be directed to another website. At this site, you will be paired with another participant. You will interpret four tasks of randomized difficulty and answer questions.

If the redirection fails to work, please return your submission, as this is indicative of JavaScript failing to work properly. If you can let us know what browser you were using, as well as any extensions you have installed when you return your submission, this will help us debug the issue. Thanks!

I.4 Study 2 Post-survey

(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 computational tasks and test contexts you saw during the study.

(1) I felt that it was **easy** to answer test **contexts**.
• Strongly disagree • Disagree • Somewhat disagree • Neither agree nor disagree • Somewhat agree • Agree • Strongly agree
(2) *(If Strongly agree, Agree, Somewhat agree selected)* What did you find to be **easy** about answering the **test contexts**?
(3) *(If Strongly disagree, Disagree, Somewhat disagree selected)* What did you find to be **difficult** about answering the **test contexts**?

This block of 9 questions was shown to all senders
The next two questions aim to understand your approach to writing instructions.

(1) What information did you make sure to **include** when **writing instructions**?
(2) Please describe how you **structured your instructions**.

These next questions relate more generally towards your experiences with the **computational tasks** (as described by the images) and **test contexts** you encountered during this study.

(1) I believe that the **computational tasks** (as described by the images) to me were **clearly understandable**.
    • Strongly disagree • Disagree • Somewhat disagree • Neither agree nor disagree • Somewhat agree • Agree • Strongly agree
(2) What about the **computational tasks** (as described by the images) made them **easy** to understand?
(3) What about the **computational tasks** (as described by the images) made them **difficult** to understand?
(4) I believe that the task **instructions I wrote** and sent the other participant were **clearly understandable**. • Strongly disagree • Disagree • Somewhat disagree • Neither agree nor disagree • Somewhat agree • Agree • Strongly agree
(5) I believe that the other participant **would have difficulty** answering test contexts correctly using my **instructions**. • Strongly disagree • Disagree • Somewhat disagree • Neither agree nor disagree • Somewhat agree • Agree • Strongly agree
(6) What did you find to be **easy** about writing the **instructions**?
(7) What did you find to be **difficult** about writing the **instructions**?

This block of 2 questions was shown to all receivers assigned to the interactive condition

(1) I believe that the **instructions** 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 **instructions** provided to me **after the other participant made any clarifications I requested**, were clearly understandable. If you did not request any clarifications, select "not applicable".
    • Strongly disagree • Disagree • Somewhat disagree • Neither agree nor disagree • Somewhat agree • Agree • Strongly agree

This block of 3 questions was shown to all receivers assigned to the non-interactive condition

(1) I believe that the **instructions** provided to me were **clearly understandable**.
    • Strongly disagree • Disagree • Somewhat disagree • Neither agree nor disagree • Somewhat agree • Agree • Strongly agree
(2) I was sometimes **confused** by the **instructions** the other participant wrote.
    • Strongly disagree • Disagree • Somewhat disagree • Neither agree nor disagree • Somewhat agree • Agree • Strongly agree
(3) What was **confusing** about the other participant's **instructions**?

This block of 3 questions was shown to all participants assigned to the interactive condition

(1) Giving or receiving feedback (including chatting if applicable, revising instructions) **improved** my understanding of the **computational task**. If you did not give or receive feedback, select "not applicable".
    • Strongly disagree • Disagree • Somewhat disagree • Neither agree nor disagree • Somewhat agree • Agree • Strongly agree
(2) *(If Strongly agree, Agree, Somewhat agree selected)* Why did giving or receiving feedback (including chatting if applicable, revising instructions) improve your understanding of the **computational task**?
(3) *(If Strongly disagree, Disagree, Somewhat disagree selected)* Why did giving or receiving feedback (including chatting if applicable, revising instructions) not improve your understanding of the **computational task**?

This block of 6 questions was shown to all senders assigned to the interactive condition

(1) Receiving feedback (including chatting if applicable, revising instructions) **improved** my **instructions**. If you did not receive feedback, select "not applicable".
    • Strongly disagree • Disagree • Somewhat disagree • Neither agree nor disagree • Somewhat agree • Agree • Strongly agree
(2) *(If Strongly agree, Agree, Somewhat agree selected)* Why did receiving feedback (including chatting if applicable, revising instructions) **help** you improve your **instructions**?
(3) *(If Strongly disagree, Disagree, Somewhat disagree selected)* Why did receiving feedback (including chatting if applicable, revising instructions) **not help** you improve your **instructions**?
(4) Giving or receiving feedback (including chatting if applicable, revising instructions) **did not** help me **avoid mistakes**. If you did not give or receive feedback, select "not applicable".
    • Strongly disagree • Disagree • Somewhat disagree • Neither agree nor disagree • Somewhat agree • Agree • Strongly agree

(5) *(If Strongly agree, Agree, Somewhat agree selected)* Why did giving or receiving feedback (including chatting if applicable, revising instructions) **help** you avoid **mistakes**?
(6) *(If Strongly disagree, Disagree, Somewhat disagree selected)* Why did giving or receiving feedback (including chatting if applicable, revising instructions) **not help** you avoid **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.
(5) How familiar are you with AI coding tools like GitHub Copilot/GitHub Copilot-X?
• Very unfamiliar • Moderately unfamiliar • Somewhat unfamiliar • Neither familiar nor unfamiliar • Somewhat familiar • Moderately familiar • Very familiar
(6) *(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
(7) Have you ever used AI coding tools like GitHub Copilot/GitHub Copilot-X?
• Yes • No

This block of five questions was shown to participants who said Yes.

(a) What AI coding tool(s) have you ever used? (e.g., GitHub Copilot, CodeLlama, ...)
(b) What AI coding tool do you use **most frequently**?
(c) For the AI coding tool you use **most frequently**, describe the task(s) you commonly use it for.
(d) For the AI coding tool you use **most frequently**, describe the specific thing(s) you think that AI coding tool is **least useful** for.
(e) For the AI coding tool you use **most frequently**, describe the specific thing(s) you think that AI coding tool is **most useful** for.

(8) How familiar are you with AI chatbots like ChatGPT or Gemini?
• Very unfamiliar • Moderately unfamiliar • Somewhat unfamiliar • Neither familiar nor unfamiliar • Somewhat familiar • Moderately familiar • Very familiar
(9) Have you ever used an AI chatbot like ChatGPT or Gemini?
• Yes • No

This block of five questions was shown to participants who said Yes.

(a) What AI chatbot(s) have you ever used? (e.g., ChatGPT, Gemini, ...)
(b) What AI chatbot do you use **most frequently**?
(c) For the AI chatbot you use **most frequently**, describe the task(s) you commonly use it for.
(d) For the AI chatbot you use **most frequently**, describe the specific thing(s) you think that AI chatbot is **least useful** for.
(e) For the AI chatbot you use **most frequently**, describe the specific thing(s) you think that AI chatbot is **most useful** for.

(10) *(Optional)* If there's anything else you'd like to tell us about this study, please enter it here.