LEETPROMPT: Leveraging Collective Human Intelligence to Study LLMs

Sebastin Santy **Ayana Bharadwaj** **Sahith Dambekodi**
Alex Albert **Cathy Yuan** **Ranjay Krishna**

Abstract

Writing effective instructions (or prompts) is rapidly evolving into a dark art, spawning websites dedicated to collecting, sharing, and even selling instructions. Yet, the research efforts evaluating large language models (LLMs) either limit instructions to a predefined set or worse, make anecdotal claims without rigorously testing sufficient instructions. In reaction to this cottage industry of instruction design, we introduce LEETPROMPT: a platform where people can interactively explore the space of instructions to solve problems. LEETPROMPT automatically evaluates human-LLM interactions to provide insights about both LLMs as well as human-interaction behavior. With LEETPROMPT, we conduct a within-subjects user study ($N = 20$) across 10 problems from 5 domains: biology, physics, math, programming, and general knowledge. By analyzing 1178 instructions used to invoke GPT-4, we present the following findings: First, we find that participants are able to design instructions for all tasks, including those that problem setters deemed unlikely to be solved. Second, all automatic mechanisms fail to generate instructions to solve all tasks. Third, the lexical diversity of instructions is significantly correlated with whether people were able to solve the problem, highlighting the need for diverse instructions when evaluating LLMs. Fourth, many instruction strategies are unsuccessful, highlighting the misalignment between participant's conceptual model of the LLM and its functionality. Fifth, participants with prompting and math experience spend significantly more time on LEETPROMPT. Sixth, we find that people use more diverse instruction strategies than these automatic baselines. Finally, LEETPROMPT facilitates a learning effect: participants self-reported improvement as they solved each subsequent problem.

1 Introduction

We are witnessing a *Cambrian explosion* of research in large language models (LLMs). LLMs have progressed from simply "understanding language" to assisting with problems in biology, solving math problems, answering general knowledge questions, and even writing code [5, 11, 43]. Given these newfound abilities, tracking how well these models work for different tasks is becoming increasingly difficult. In response, large benchmarking attempts, including GLUE [61], SuperGLUE [60], BIG-Bench [20], and HELM [31], have been proposed. However, we argue that they remain limited since they evaluate models using a fixed set of LLM "instructions", or what we often colloquially refer to as "prompts" [31]. This limitation is highlighted by recent manuscripts that break away from such benchmarks and resort to showcasing LLM capabilities (e.g. specifically GPT-4's) using anecdotal interactions between researchers exploring the space of instructions [6, 42].

While the need to study how people invoke LLMs has always been at the center of today's discourse, in the context of human-AI alignment [12, 16], there is a dearth of non-anecdotal evaluations of LLMs with real human interactions. Such studies are particularly vital since the difficulty of finding effective instructions has led to websites and forums dedicated to collecting and sharing instructions (e.g.

Preprint. Under review.

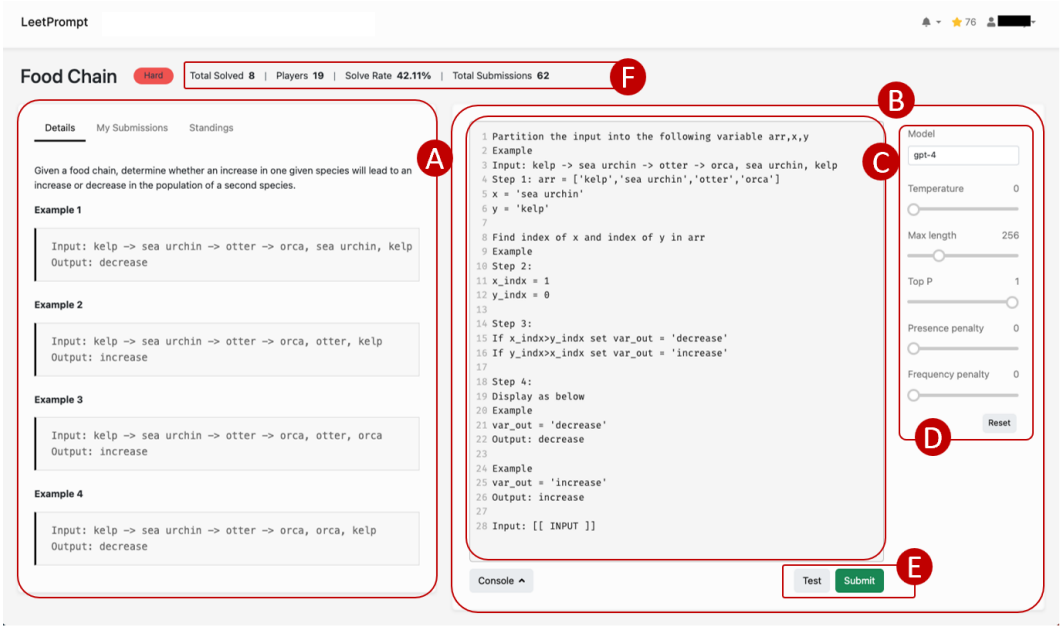


Figure 1: LEETPROMPT is a platform where users can explore the space of instructions to solve tasks with LLMs. **A.** Problem description: This panel contains a description of the problem that needs to be solved. It often contains examples of plausible inputs and outputs. **B.** Interaction interface: Here, users will write instructions, change model hyper-parameters, and evaluate their instructions against public as well as private test cases. **C.** Writing instructions: Here lies the text interface where users write their instructions. The token `[[ INPUT ]]` identifies where test cases will insert inputs. Most problems already contain a starter instructions to help users get started. **D.** Model hyper-parameters: This panel shows which model is being used and the corresponding modifiable model hyper-parameters. **E.** Test and submit: Users can test their instructions against a custom input by clicking Test or submit their instructions to be evaluated against the blind test cases by clicking on Submit. **F.** Problem details and submissions: Users can check overall problem statistics, load past submissions, and compare their performance against other users.

PromptHero, Arthub.ai, Reddit/StableDiffusion). There are also online marketplaces for purchasing and selling useful instructions (e.g. PromptBase).

To enable a rigorous evaluation with real human interactions, we present LEETPROMPT. LEETPROMPT is an online platform populated with problems that users can attempt to solve by invoking LLMs with custom instructions; it inherits its name from LeetCode, where users can attempt to solve similar problems by writing custom code (Figure 1). LEETPROMPT is a dual objective platform similar to re-CAPTCHA [59] and other collective human protocols [24]: On one hand, users use LEETPROMPT to solve problems with LLMs; meanwhile, LEETPROMPT automatically gathers evaluation metrics and user-behavior insights for researchers. On LEETPROMPT, users also have the ability to add new problems, allowing the platform to organically grow. By granting people agency, our platform leverages collective human intelligence [33] to study which problems are unsolvable by language models and simultaneously provides user statistics on their interaction behavior.

To showcase the utility of LEETPROMPT, we run a within-subjects evaluation where we invite participants to solve problems spanning 5 domains: Biology, Physics, Math, Programming, and General Knowledge. We chose these domains as representative of current LLM evaluations [22, 40, 48, 53, 49, 36, 54, 8, 32]. We invited a set of 4 initial users with prior experience evaluating and interacting with LLMs to write problems for LEETPROMPT. They were also asked with problem setting: generating a set of problems and their corresponding private test cases, where the test cases serve the same purpose as those used in software development. In total, the problem setters populated LEETPROMPT with 101 questions. From this set of questions, we sample 10 questions for our user study, such that there are 2 questions per domain. We recruit 20 participants, with varying education background, experience with LLMs/programming, and demographics. Each participant attempts to solve the 10 sampled problems using GPT-4.

By quantitatively analyzing and qualitatively coding the interactions across 1178 invocations of GPT-4, we make the following observations: First, we find that participants are able to design instructions for all tasks, including those that problem setters deemed unlikely to be solved. Sometimes, incorrect instructions can still result in correct LLM behavior. Second, all automatic mechanisms fail to generate instructions to solve all tasks, including zero-shot, few-shot, zero-shot CoT [26], few-shot CoT [64], and auto-CoT [70]. Third, the lexical diversity of instructions is significantly correlated with people's ability to solve the problem, highlighting the need for diverse instruction strategies. Fourth, many instruction strategies are unsuccessful, highlighting the misalignment between the participant's conceptual model of the LLM and its functionality. Fifth, participants with experience on instructing language models, and with math spend significantly more time on LEETPROMPT. Sixth, we find that people use more diverse instruction strategies than these automatic baselines. Finally, LEETPROMPT facilitates a learning effect: participants self-report solving subsequent problems faster and with a better strategy.

2 Related Work

We design LEETPROMPT by drawing on ideas from existing benchmarks, recent anecdotal evaluations, and collective intelligence literature.

Benchmarks in machine learning. Reaching human performance on influential benchmarks is often viewed as a key milestone for a field [46]. For instance, AlphaFold's superior performance on the CASP 14 competition marks a major scientific advance in the field of structural biology. Early benchmarks in machine learning include Switchboard [21] and MNIST [28], which have been followed by ImageNet [47], SQuAD [44], and SNLI [3]. These benchmarks have rallied computer vision, natural language processing, and other domains of machine learning around a set of common goals. With the onset of LLMs, larger benchmarks such as GLUE [61, 60], Eval Harness [19], BigBench [20], MMLU [22], and HELM [31] have helped define progress.

A trend towards anecdotal evaluation. Moving away from benchmarks, anecdotal proof of these models' newfound powers is one of the most recent trends. "Sparks of AGI", for example, shows numerous examples suggesting that GPT-4 pushes the needle towards artificial general intelligence [6]. Many other papers have followed a similar pattern reporting on manually curated data points exhibiting surprising performance of language models [42] and laying big claims such as language models possessing a theory of mind [27]. Without sufficient test cases for each claim and without an exhaustive exploration of possible input instructions, these claims require further investigation [64, 26]. LEETPROMPT serves as a platform where researchers can design problems, generate sufficient test cases, and allow a collective of users to explore the space of instructions.

Collective intelligence systems. Leveraging people's collective intelligence has long since served the machine learning community. Collective intelligence, in the form of micro-task crowdsourcing, catalyzed the community's ability to annotate training [47] and curate evaluation data [72]. Dual objective platforms, such as re-CAPTCHA [59], simultaneously serve as a spam filter while also passively collecting annotations for vision and language tasks. Games-with-a-purpose is another class of systems where players derive casual enjoyment while playing games that contribute useful data for image labeling [57], image segmentation [58], commonsense reasoning [56], math (Guess the correlation) and even protein folding (FoldIt). Lab-in-the-Wild projects similarly provide users with results from a psychology test, exchanging curiosity for their efforts [45]. Citizen science projects such as Stardust [65] and Zooniverse [51] host projects in a range of fields including astronomy, ecology, cell biology, humanities, and climate science. We draw inspiration from these methods to design LEETPROMPT's dual use of LLMs evaluation while also studying user interaction behavior.

Human-driven evaluation. Recent work advocates for evaluating LLMs with human interactions [29]. While they primarily leverage human interactions to analyze LLM outputs [29], we propose the opposite: we explore human interactions with LLM input instructions. Today, human-driven evaluations determine whether text is human- or machine-generated [14] and measure how well generated images reflect human text inputs [25] [1]. Corporations have also started efforts into outsourcing their "red-teaming" evaluation to people in the wild [17, 39]. Our contribution can also be seen as a mirror opposite of Dynabench [24], which incentivizes users to provide adversarial

[1] https://artwhisperer.io/

data to break model behavior; LEETPROMPT users produce instructions to most effectively solve problems.

3 LEETPROMPT

Our research introduces LEETPROMPT: an online platform where users are presented with problems from multiple domains and tasked with writing instructions that an LLM can use to solve the task. We hope to release LEETPROMPT publicly at a later date to test its utility at scale.

Designing the LEETPROMPT workflow. LEETPROMPT's core functionality is inspired by online code judging systems such as LeetCode, TopCoder, and CodeForces. The platform contains a list of problems that users choose to tackle. Each problem also has a leaderboard indicating how many users have already attempted to tackle this problem, who achieved the best performance, and with how many submissions. Once the user chooses a problem, they are given the problem's description as well as a few examples of inputs and outputs outlining expected behavior. With this information, users must write instructions that steer a large language model toward solving the given problem. Users can also change the model's hyperparameters.

Once they've finished writing the instruction, users can test its utility against their own custom inputs. Custom inputs allows users to iteratively explore how their changes affect LLM behavior. Once satisfied, they can submit their final instruction to be evaluated against our hidden test cases. These private test cases are never revealed to the user; instead, they receive a percentage score indicating how many of the private test cases passed. To discourage rapid-fire submission and hill-climbing behavior on the private test cases, the platform provides a time-out after 3 submissions for 5 minutes. LEETPROMPT also records how many times the user submits instructions, and uses that information to rearrange the leaderboard to tacitly encourage fewer submissions.

Designing the LEETPROMPT's interface. LEETPROMPT's interface is designed to be simple and promote an intuitive user experience (see Figure 1). We recruit UX designers to join our team and run studies using mockups of the interface to identify common pitfalls. We face the following design challenge: creating a platform that is intuitive for users who are not familiar with the process of writing instructions for LLMs. Drawing on design theory [38], the following principles are used: (1) Standardizing the interface to match with existing/popular LLM platforms and (2) employing recognition over recall so users know the full extent of the platform's affordances through a quick skim. The appendix contains older versions of the interface.

Evaluating user-generated instructions. Figure 1(C) contains a screenshot of a possible instruction that a user might write to solve a problem. Users are expected to place an `[[ INPUT ]]` token somewhere in their produced instruction. LEETPROMPT evaluates instructions using the following:

$$\text{accuracy} = 100 \times \frac{1}{N} \sum_{i=1}^{N} \mathbb{1}[y_i == LLM((\text{Instruction; } \texttt{[[INPUT]]} = x_i))]$$

where $(x_i, y_i) \in [1, \ldots, N]$ are N private test case (input, output) pairs. (Instruction; `[[ INPUT ]]` $= x_i$) replaces the input token in the instruction with test case input x_i. $\mathbb{1}[\cdot]$ is the indicator function. LEETPROMPT contains a set of common regexes to extract the desired output for the problem from the LLM's generation.

4 User study

In the following section, we outline our user study design in which we study how people interact with LLMs to solve problems.

Domain selection. Language models were tested on several different problems. Pre-LLMs, most language systems focused on tasks such as sentiment analysis [34], part-of-speech tagging [10], machine translation [52], entailment [35], and speech processing [37]. Since LLMs, models are now evaluated on simple mathematical and commonsense reasoning [22, 40, 48, 53], causal reasoning [23] and theory of mind [49, 36], basic sciences [54], programming [8, 32], and even law exams [71, 39]. We choose a subset of these domains as the test bed for our user study: Biology, Physics, Math, Programming, and General Knowledge.

Table 1: User performance across the 10 questions in 5 domains. **Difficulty** of the question was pre-determined by the problem setters. **#Int** is the number of model interactions, **#Sub** is the number of instructions that were submitted, **Passed** is the number of passed test cases, and **Solved** is the percentage of participants who able to successfully solve the problem. **LD**, **SD**, and **AD** are the Lexical, Semantic, and Approach Diversity, respectively.

Domain	Question	Difficulty	#Int	#Sub	Passed	Solved	LD	SD	AD
Biology	Water Potential	Easy	30	18	4.72 ± 0.83	94%	0.56	**3e-02**	0.61
Biology	Food Chain	Hard	76	45	3.07 ± 1.47	50%	0.48	6e-03	0.49
Physics	Ideal Gas Law	Easy	108	31	3.68 ± 1.76	88 %	0.38	**7e-02**	0.25
Physics	Resistance is Futile	Medium	84	48	2.88 ± 1.76	50%	0.43	8e-02	0.55
Math	Consecutive Integers	Easy	67	25	4.36 ± 1.66	75%	**0.68**	5e-03	**0.80**
Math	Digit Sum	Hard	134	45	3.27 ± 0.75	6 %	0.46	5e-03	0.58
Programming	Intersperse	Medium	99	23	3.70 ± 1.46	63%	0.51	8e-03	0.74
Programming	Sort Numbers	Medium	64	27	3.56 ± 1.53	56%	0.49	2e-03	0.68
Knowledge	Beatles Title	Medium	177	28	3.36 ± 1.57	38%	0.47	4e-04	0.44
Knowledge	Theory of Mind	Medium	36	13	3.15 ± 2.23	44%	0.39	3e-04	0.60
	Average $\rightarrow$		87.5	30.3	3.58 ± 0.55	56%	0.49	2e-02	0.57

Problem setters. LEETPROMPT grants people agency by empowering them to use our platform to study new problems they might struggle to solve with LLMs. We invite a small set of 4 initial users with prior experience interacting with and using LLMs to LEETPROMPT We ask these users to design problems: they collectively write a set of 101 problems, each with a set of 5 private test cases. Once the problems are designed, the problem designers are asked to try and solve each problem and to assign a subjective difficulty based on their experience. Each problem is categorized into one of three difficulty levels using majority vote. Easy problems are those that can be solved by utilizing a simple formula or equation. Medium problems contain difficulty either in the complexity of the formulas or parsing the inputs and describing the task. They sometimes require domain knowledge or some background information that can simplify the problem. Hard problems involve complex logic or edge cases.

From this set, we sample 10 problems such that there are 2 problems for each of the 5 aforementioned domains. Each participant in our user study sees these 10 problems. Table 1 describes these 10 problems. In total there are 3 easy problems, 5 medium problems and 2 hard problems. The 2 hard problems were chosen because the problem setters were not able to find an instruction that worked for all the test cases. Problems are randomly ordered for each user.

Example problem. Figure 1 lays out one of our Biology problems and a user-generated instruction. It presents the participant with a food chain in the form of $X \rightarrow Y$ implies that species Y is at a higher tropic level and, therefore, eats species X. The user is also presented with two species sampled from that food chain. The problem asks the user to determine whether an increase in the population of the first species will lead to an increase or decrease in the second. Example 1 in the problem description presents the following food chain: kelp $\rightarrow$ sea urchin $\rightarrow$ otter $\rightarrow$ orca. The two species presented are "sea urchin" and "kelp". Since "sea urchins" eats kelp, an increase in the "sea urchin" population leads to a decrease in the "kelp" population. This is a hard problem.

The user-generated instruction uses pseudocode to inform the model that if the first species is at a higher trophic level than the second, the population of the second species will decrease, and vice versa. Even though this logic works for "sea urchin" and "kelp", this pseudocode is incorrect when the two species are "otter" and "kelp". Since there is a skip connection between "otter" and "kelp", an increase in "otters" will result in a decrease in "sea urchins", which will in turn result in an increase in "kelp".

Study protocol. We design a within-subjects study to explore the utility of LEETPROMPT. To reiterate, the study measures LEETPROMPT's utility along two simultaneous goals: Investigating 1) whether the platform supports participants in writing effective instructions to solve problems with LLMs and 2) providing insights into how people interact with LLMs as they explore the space of instructions.

The study begins by asking participants to sign a consent form that outlines that their interactions will be stored and used for this research contribution. Next, they are asked an initial survey to record their demographics, their background education, and their experience working with LLMs or programming more broadly. Next, they are shown instructions for using the interface and provided resources that

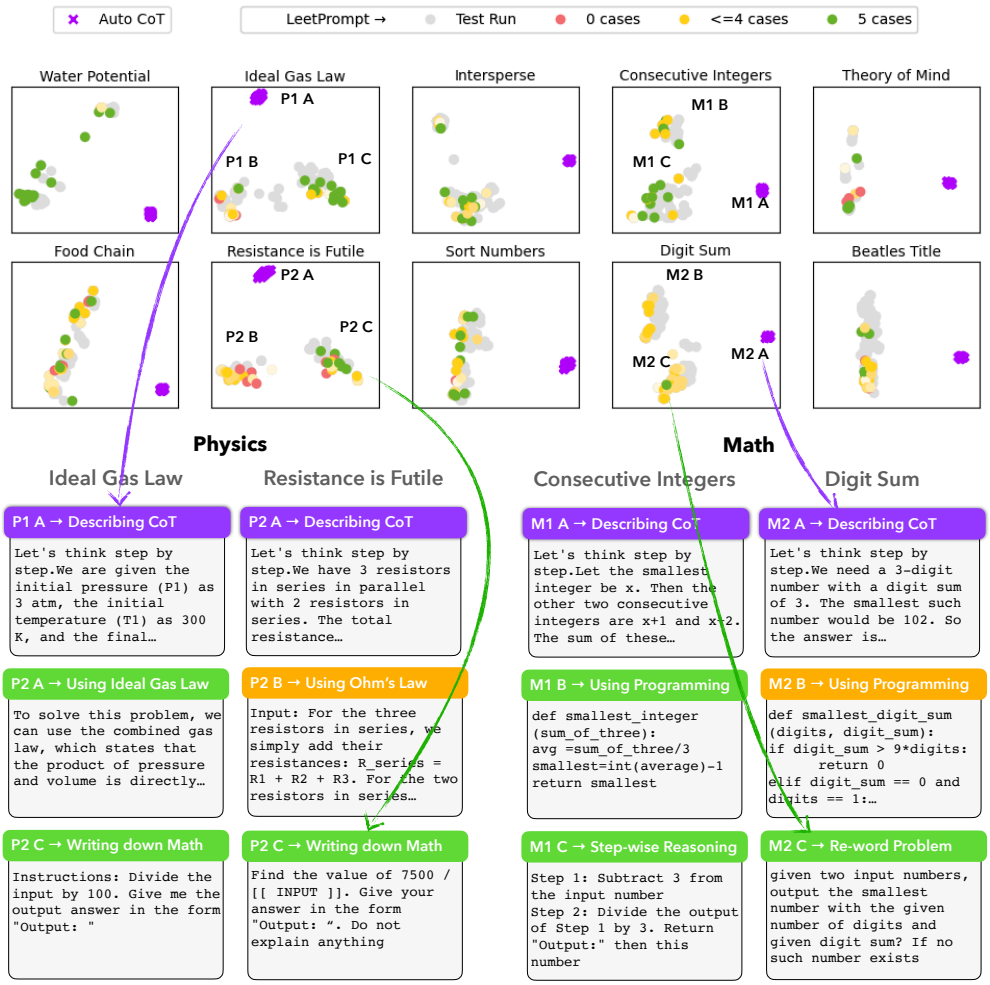


Figure 2: **Visualizing the space of Auto-CoT and LEETPROMPT Instructions**: 2D Principal Component Analysis (PCA) of embeddings of Auto-CoT and collected instructions from LEETPROMPT. Auto-CoT instructions are marked as purple X. LEETPROMPT instructions are marked as dots with colors representing its solvability: "Test" instructions are colored gray. For "Submitted" instructions, red color indicates that they failed all testcases, yellow indicates they passed 1-4 testcases, and green indicates that they passed 5 testcases. Instructions are specifically shown for two problems each from each domains to illustrate the different strategies used by participants and whether they were successful.

they can reference for advice on good instruction design. After this, the participant is shown a demo of the interface, which highlights and explains the various panes in the interface. To help familiarize themselves with the interface, users are provided a sandbox interface with a toy starter problem that they can test the functionality afforded by the platform. Next, the participants are presented with the 10 sampled problems and asked to attempt solving each one. We set expectations that the study will likely take about 1 hour. The study ends with a final survey where we collect open-ended responses describing their experience.

Participant selection. All of our participants are recruited locally from the authors' city/town. We only include participants that speak English, spent at least 1 hour working on problems, and submitted at least one instruction to each of the 10 problems. All participants receive a payment of $15 for their time with no bonuses. We provide no other incentives other than intrinsic motivation for subsequent attempts at designing better instructions for each problem. Regardless, we find that each user tried an average of 5.8 instructions per question.

Measured variables. To evaluate instructions, we use two primary objective performance measures: solvability and diversity. *Solvability* measures the percentage of test cases that pass when using a

specific instruction. *Diversity* measures how creative people are in writing instructions. Diversity provides insights into the different approaches that a participant takes to solve each problem. It allows us to evaluate the claim that diverse instructions and ensembling leads to significant boosts in LLM performance [67]. Since diversity of instructions can be measured in multiple ways, we measure the following: (1) *Lexical Diversity (LD)*, *Semantic Diversity (SD)*, and *Approach Diversity (AD)*.

LD uses Repetition Rate (RR) [7, 2] to measure diversity in lexicons between a participant's instruction and the original wording used in the problem description. Specifically, LD $= 1 -$ RR. LD is 0 if all lexicons appear more than once and 1 if all lexicon are new.

SD measures the diversity in strategies used by a user when tackling a specific problem. We use text+code embeddings (text-embedding-ada-002) to encode each instruction and visualize their first two principle components for qualitative visual analysis. For a quantitative analysis, SD is the variance for the first principal component.

AD measures the percentage of instruction strategies used, where strategies are qualitatively coded by 3 independent coders (see Table **??**) for codes and appendix for full details). The qualitative codes reflect common instruction strategies in literature: prime the model to behave like a domain expert [1], use example input/outputs [62], use CoT prompting [64, 26, 63, 66, 55], use self-help [41, 68] and the lexical structure [15], and use pseudo-code [69].

Automatic instruction mechanisms. We compare user-generated instructions with mechanisms used in existing benchmarks. Zero-shot (*0s*) uses the problem description and no other information. Zero-shot CoT (*CoT*) includes a phrase which requests the LLM to reason the steps aloud [26]. Few-shot variants (N-shot *Ns* where $N = 1, 2, 3, 4$) append N-example input/output pairs. We also test with advanced auto-prompting methods, such as *Auto-CoT* [70] which invokes multiple CoT reasoning steps along with 10 input/output examples, and Synthetic prompting [50] which tunes existing prompts.

Models used. LEETPROMPT supports any LLM that affords an API access to invoke the model. We allow participants to use any of the following models: GPT-4, GPT-3.5, and GPT-3. However, all participants chose to use GPT-4 for all their attempts. In the appendix, we retroactively evaluate their instructions on the other models as a point of comparison.

5 Results

By analyzing the 1178 user-generated instructions and their feedback from the survey we find the following: To start, participants are able to generate instructions to solve all the problems, including the hard questions that the problem setters had deemed unlikely to be solved. Sometimes, incorrect instructions can still result in correct LLM behavior. Second, automatic mechanisms are not able to solve all problems. Third, the use of diverse instructions is strongly correlated with solvability, indicating a need for diversity in instructions in existing benchmarks. Fourth, there is a misalignment between user expectations of LLMs and the reality of how interpreted user instructions; participants stated that they were unprepared for the challenges they faced. Fifth, participants with more experience in prompting or math spent more time on LEETPROMPT. Sixth, we find that people uncovered a diverse array of instruction strategies than these automatic baselines. Finally, participants self-reported a learning effect and better strategy use as they progressed through the study.

Participants found solutions even though the problem setters were not able to solve them. As shown in Table 1, all problems were solved by at least one participant. The problem setters' difficulty categorization is strongly correlated with how many test cases participants passed ($r(19) = .74, p = .01$) and what percentage of participants were able to solve all 5 private test cases ($r(19) = 0.84, p = .001$). For "Digit Sum", only one participant was able to solve the question. Surprisingly, the successful solution involved a specific re-wording of the question that improved the problem's clarity and thus made it easier for the model to understand and solve the problem. Only through collection action is LEETPROMPT able to identify this solution. Similarly, half the participants were able to solve "Food Chain". Surprisingly, **one winning instruction was a logically incorrect reasoning step that somehow still passes the test cases** (shown in Figure 1). This adds more support to concurrent work, announced this week, which also finds that unfaithful reasoning steps improve LLM performance [55].

Table 2: Comparison of existing prompting approaches to prompts collected from LEETPROMPT. **0s**: Zero-shot; **0s CoT**: Zero-shot Chain-of-Thought prompting; **1s, 2s, 3s, 4s**: 1,2,3,4 shot (or examples) prompting; **Auto-CoT**: Automatic Prompting Method; **Ours**: Prompts from LEETPROMPT. **P** denotes the maximum number of testcases that the the given method was able to pass. As Auto-CoT and Ours have multiple prompts per problem, we report **LD** & **CD** which are the lexical and content diversity of prompts for each problem. We do not report **AD** for Auto-CoT since it defaults to using CoT as the main strategy for solving the problem.

		0s CoT		**1s**	**2s**	**3s**	**4s**	**Auto-CoT**			**Ours**		
Domain	**Question**	**P**	**P**	**P**	**P**	**P**	**P**	**P**	**LD**	**SD**	**P**	**LD**	**SD**
Biology	Water Potential	2	2	3	**5**	**5**	**5**	**5**	0.17	2e-05	**5**	0.56	3e-02
Biology	Food Chain	4	3	3	3	2	3	**5**	0.25	3e-05	**5**	0.48	6e-03
Physics	Ideal Gas Law	**5**	3	4	4	4	**5**	4	0.42	1e-04	**5**	0.38	7e-02
Physics	Resistance is Futile	0	0	3	3	4	2	3	0.42	2e-04	**5**	0.43	8e-02
Math	Consecutive Integers	**5**	4	**5**	4	4	**5**	**5**	0.34	2e-05	**5**	0.68	5e-03
Math	Digit Sum	3	2	3	3	3	4	**5**	0.31	2e-05	**5**	0.46	5e-03
Programming	Intersperse	**5**	**5**	**5**	**5**	**5**	**5**	4	0.10	2e-05	**5**	0.51	8e-03
Programming	Sort Numbers	0	0	4	4	3	3	4	0.37	9e-05	**5**	0.49	2e-03
Knowledge	Beatles Title	**5**	**5**	4	4	4	**5**	**5**	0.11	5e-05	**5**	0.47	3e-04
Knowledge	Theory of Mind	0	0	**5**	**5**	**5**	**5**	**5**	0.11	3e-05	**5**	0.49	4e-04
		29	24	39	40	39	42	45	0.26	6e-05	**50**	**0.49**	**2e-02**

Table 3: Pearson's correlation coefficient between participant attributes (demographic, background, and experience) and the maximum number of test cases passed and the time taken for each problem. '..' indicates trending towards significance ($p < 0.1$) and '*' denotes significance ($p < 0.05$). **Pass** is the average number of testcases passed and **Time** is the avg. time taken between first and last interaction with the problem

Domain →	**Biology**		**Physics**		**Math**		**Programming**		**General**		**Overall**	
Participant ↓	**Pass**	**Time**	**Pass**	**Time**	**Pass**	**Time**	**Pass**	**Time**	**Pass**	**Time**	**Pass**	**Time**
Demographic												
Age	0.23	-0.23	-0.13	0.20	0.13	0.29	0.36	0.44..	0.13	0.23	0.21	0.32
Experience												
Biology	-0.03	-0.11	-0.32	0.03	-0.41..	-0.50*	-0.26	0.16	-0.29	-0.09	-0.37	-0.24
Physics	0.12	-0.01	0.18	0.46..	0.04	0.03	0.21	-0.08	0.31	-0.23	0.25	0.02
Math	0.18	0.25	0.01	0.31	0.10	0.30	0.34	0.41..	0.25	0.27	0.26	0.54*
Trivia	-0.01	0.19	0.08	-0.11	0.25	-0.05	0.45..	0.34	0.22	0.01	0.30	0.11
LLM	0.09	0.51*	0.14	0.22	0.43..	0.19	0.36	0.25	0.43..	0.42..	0.42..	0.59*
Prompting	0.43..	0.13	0.21	0.54*	0.35	0.16	0.35	0.06	0.25	0.35	0.43..	0.45..
Programming	-0.27	0.18	-0.05	-0.38	0.09	-0.22	0.19	0.15	0.25	0.19	0.10	-0.02

None of the automatic mechanisms were able to find instructions to solve all the problems (Table 2). Furthermore, the diversity metrics (LD and SD) are both significantly smaller. A low LD implies that these mechanisms do not deviate from the original problem description. We visualize the lack of diversity of Auto-CoT in Figure 2, which visualizes the first two principal components of the SD instruction embeddings for each problem.

Lexical diversity of instructions is significantly correlated with the number of passed test cases ($p(19) = 0.71, p < 0.01$). (Table **??**). This mirrors prior work [30] which found that having diverse reasoning allowed for better LLM performance. This result suggests that large LLM benchmarks should rethink the use of a small fixed set of instructions. From Table 2 the best automatic mechanism, Auto-CoT, has lower LD than LEETPROMPT participants (0.26 vs 0.49) and also passes fewer number of test cases (45 vs 50) which further proves this point.

Participants struggled when debugging unexpected model behavior. Participants reported a sense of confusion between their expectations versus how the model worked. Figure 2 shows visually that there exists entire clusters of instructions that do not solve the problem. For example, math-related strategies to solve the two math problems didn't work while programming-related strategies did. Participants complained that the model "would do math incorrectly" (P42). Similarly, using domain-specific information, such as using Ohm's Law to solve the "Resistance is Futile" Physics question failed while using math-related instructions sometimes succeeded. Even so, one participant exclaimed, "what was also strange was that the numerical answer the LLM provided could change based on seemingly trivial additions to the prompt: "I would perturb prompts in small ways that might unexpectedly change the output" (P37).

Participants with experience with LLMs, or an education in math spent significantly more time writing instructions ($p(19) = 0.59, p < 0.05$) and ($p(19) = 0.54, p < 0.05$). From Table 4, those with either experience with LLMs or prompting were trending towards significance for solving more problems ($p(19) = 0.42, p < 0.1, p(19) = 0.43, p < 0.1$). From Table 4, **there is no strong correlation between the participant's domain knowledge with solving problems in that domain**.

Taken together, these two findings suggest that knowing how to instruct the model can be more important than domain knowledge about the problem.

Participants uncovered a variety of instruction strategies to solve problems. Each column in Figure 2 is a different domain. Interestingly, the clusters appear to follow a similar pattern between the two rows, implying that people use similar strategies for problems within a given domain.

On average, natural language worked better than pseudocode. By qualitatively analyzing the clusters in Figure 2), we find that participants who used pseudocode instructions generally wrote longer instructions to account for all possible edge cases, similar to writing exceptions in software. Debugging these instructions was easier even though writing the complex instruction took longer. For those who wrote natural language instructions, their instructions were shorter, and more successful. The one exception was the Physics domain, for which simplifying the domain-specific knowledge about physics into arithmetic equations worked the best. Unfortunately, participants with a background in programming mentioned that they found it difficult to break away from writing programming instructions, "Was thinking in a programming language/code and found it hard to translate that to an instruction" (P57) and, "Wasn't able to express programmatic reasoning" (P42).

Explicit, detailed instructions do better but instruction structure can also affect performance. Participants learned a need for clarity and to "give more direct and structured commands" (P34) and be "extremely explicit and instructive" (P48) in order for the model to understand. Others noticed that relatively trivial changes led to different results in unpredictable ways, such as "'do not include any explanation and just provide the answer' changing the answer" (P48). Related, in "Beatles Title", we notice that the approaches did not cluster into easily separable clusters (Figure 2). This is because the question purely relied on factual knowledge and did not require explicit reasoning. In the absence of a strategy, we evaluated how minor changes in instructions might result in better performance, finding that generous use of line breaks between examples resulted in better performance. Our future work will evaluate the generality of this finding.

Participants demonstrated and self-reported a learning effect as they progressed through the study. Multiple participants mentioned that they began adopting specific strategies as they completed more questions. "Yes, as I went through the problems I learned how to ask the model questions so it could help me answer the question. I also learned how to provide reasoning for the examples." (P39) "I realized simplifying the problems greatly increased the reliablility of the model, so I tried to rephrase questions to be as simple as possible [for future problems]" (P42).

6 Future directions and limitations

In the future, LEETPROMPT could serve purposes beyond evaluation. It could support users produce tutorials to teach others how to use LLMs; it can serve as a platform for the public to audit released models; it could also become integrated with existing benchmarks, like HELM, to provide micro-analysis evaluation of individual inputs; it can also track interaction patterns between people and LLMs over time. However, it does have its set of limitations: First, the user interactions from our study are unlikely to be indicative of LEETPROMPT's usage in the wild with 20 participants with monetary compensation. Second, the demographics of participants mostly included people who have STEM knowledge and had at least heard of language models. Third, our problems did not explore open-ended tasks, creativity tasks, or complex tasks that require tool-usage.

References

[1] L. P. Argyle, E. C. Busby, N. Fulda, J. Gubler, C. Rytting, and D. Wingate. Out of one, many: Using language models to simulate human samples. *arXiv preprint arXiv:2209.06899*, 2022.

[2] N. Bertoldi, M. Cettolo, and M. Federico. Cache-based online adaptation for machine translation enhanced computer assisted translation. In *Proceedings of the MT Summit XIV*, pages 35–42. 2013.

[3] S. R. Bowman, G. Angeli, C. Potts, and C. D. Manning. A large annotated corpus for learning natural language inference. *arXiv preprint arXiv:1508.05326*, 2015.

[4] J. Brooke et al. Sus-a quick and dirty usability scale. *Usability evaluation in industry*, 189(194):4–7, 1996.

[5] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.

[6] S. Bubeck, V. Chandrasekaran, R. Eldan, J. Gehrke, E. Horvitz, E. Kamar, P. Lee, Y. T. Lee, Y. Li, S. Lundberg, et al. Sparks of artificial general intelligence: Early experiments with gpt-4. *arXiv preprint arXiv:2303.12712*, 2023.

[7] M. Cettolo, N. Bertoldi, and M. Federico. The repetition rate of text as a predictor of the effectiveness of machine translation adaptation. In *Proceedings of the 11th Conference of the Association for Machine Translation in the Americas: MT Researchers Track*, pages 166–179, 2014.

[8] M. Chen, J. Tworek, H. Jun, Q. Yuan, H. P. de Oliveira Pinto, J. Kaplan, H. Edwards, Y. Burda, N. Joseph, G. Brockman, A. Ray, R. Puri, G. Krueger, M. Petrov, H. Khlaaf, G. Sastry, P. Mishkin, B. Chan, S. Gray, N. Ryder, M. Pavlov, A. Power, L. Kaiser, M. Bavarian, C. Winter, P. Tillet, F. P. Such, D. Cummings, M. Plappert, F. Chantzis, E. Barnes, A. Herbert-Voss, W. H. Guss, A. Nichol, A. Paino, N. Tezak, J. Tang, I. Babuschkin, S. Balaji, S. Jain, W. Saunders, C. Hesse, A. N. Carr, J. Leike, J. Achiam, V. Misra, E. Morikawa, A. Radford, M. Knight, M. Brundage, M. Murati, K. Mayer, P. Welinder, B. McGrew, D. Amodei, S. McCandlish, I. Sutskever, and W. Zaremba. Evaluating large language models trained on code. 2021.

[9] W. Chen, X. Ma, X. Wang, and W. W. Cohen. Program of thoughts prompting: Disentangling computation from reasoning for numerical reasoning tasks. *arXiv preprint arXiv:2211.12588*, 2022.

[10] A. Chiche and B. Yitagesu. Part of speech tagging: a systematic review of deep learning and machine learning approaches. *Journal of Big Data*, 9(1):1–25, 2022.

[11] A. Chowdhery, S. Narang, J. Devlin, M. Bosma, G. Mishra, A. Roberts, P. Barham, H. W. Chung, C. Sutton, S. Gehrmann, et al. Palm: Scaling language modeling with pathways. *arXiv preprint arXiv:2204.02311*, 2022.

[12] B. Christian. *The alignment problem: Machine learning and human values*. WW Norton & Company, 2020.

[13] J. Coda-Forno, K. Witte, A. K. Jagadish, M. Binz, Z. Akata, and E. Schulz. Inducing anxiety in large language models increases exploration and bias. *arXiv preprint arXiv:2304.11111*, 2023.

[14] L. Dugan, D. Ippolito, A. Kirubarajan, S. Shi, and C. Callison-Burch. Real or fake text?: Investigating human ability to detect boundaries
between human-written and machine-generated text. In *Proceedings of the 2023 AAAI Conference on Artificial Intelligence*, 2023.

[15] Y. Fu, H. Peng, A. Sabharwal, P. Clark, and T. Khot. Complexity-based prompting for multi-step reasoning. *arXiv preprint arXiv:2210.00720*, 2022.

[16] I. Gabriel. Artificial intelligence, values, and alignment. *Minds and machines*, 30(3):411–437, 2020.

[17] D. Ganguli, L. Lovitt, J. Kernion, A. Askell, Y. Bai, S. Kadavath, B. Mann, E. Perez, N. Schiefer, K. Ndousse, et al. Red teaming language models to reduce harms: Methods, scaling behaviors, and lessons learned. *arXiv preprint arXiv:2209.07858*, 2022.

[18] L. Gao, A. Madaan, S. Zhou, U. Alon, P. Liu, Y. Yang, J. Callan, and G. Neubig. Pal: Program-aided language models. *arXiv preprint arXiv:2211.10435*, 2022.

[19] L. Gao, J. Tow, S. Biderman, S. Black, A. DiPofi, C. Foster, L. Golding, J. Hsu, K. McDonell, N. Muennighoff, J. Phang, L. Reynolds, E. Tang, A. Thite, B. Wang, K. Wang, and A. Zou. A framework for few-shot language model evaluation, Sept. 2021.

[20] A. Ghazal, T. Rabl, M. Hu, F. Raab, M. Poess, A. Crolotte, and H.-A. Jacobsen. Bigbench: Towards an industry standard benchmark for big data analytics. In *Proceedings of the 2013 ACM SIGMOD international conference on Management of data*, pages 1197–1208, 2013.

[21] J. J. Godfrey, E. C. Holliman, and J. McDaniel. Switchboard: Telephone speech corpus for research and development. In *Acoustics, Speech, and Signal Processing, IEEE International Conference on*, volume 1, pages 517–520. IEEE Computer Society, 1992.

[22] D. Hendrycks, C. Burns, S. Basart, A. Zou, M. Mazeika, D. Song, and J. Steinhardt. Measuring massive multitask language understanding. *arXiv preprint arXiv:2009.03300*, 2020.

[23] E. Kıcıman, R. Ness, A. Sharma, and C. Tan. Causal reasoning and large language models: Opening a new frontier for causality. *arXiv preprint arXiv:2305.00050*, 2023.

[24] D. Kiela, M. Bartolo, Y. Nie, D. Kaushik, A. Geiger, Z. Wu, B. Vidgen, G. Prasad, A. Singh, P. Ringshia, et al. Dynabench: Rethinking benchmarking in nlp. *arXiv preprint arXiv:2104.14337*, 2021.

[25] Y. Kirstain, A. Polyak, U. Singer, S. Matiana, J. Penna, and O. Levy. Pick-a-pic: An open dataset of user preferences for text-to-image generation. *arXiv preprint arXiv:2305.01569*, 2023.

[26] T. Kojima, S. S. Gu, M. Reid, Y. Matsuo, and Y. Iwasawa. Large language models are zero-shot reasoners. *arXiv preprint arXiv:2205.11916*, 2022.

[27] M. Kosinski. Theory of mind may have spontaneously emerged in large language models. *arXiv preprint arXiv:2302.02083*, 2023.

[28] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[29] M. Lee, M. Srivastava, A. Hardy, J. Thickstun, E. Durmus, A. Paranjape, I. Gerard-Ursin, X. L. Li, F. Ladhak, F. Rong, et al. Evaluating human-language model interaction. *arXiv preprint arXiv:2212.09746*, 2022.

[30] Y. Li, Z. Lin, S. Zhang, Q. Fu, B. Chen, J.-G. Lou, and W. Chen. On the advance of making language models better reasoners. *arXiv preprint arXiv:2206.02336*, 2022.

[31] P. Liang, R. Bommasani, T. Lee, D. Tsipras, D. Soylu, M. Yasunaga, Y. Zhang, D. Narayanan, Y. Wu, A. Kumar, et al. Holistic evaluation of language models. *arXiv preprint arXiv:2211.09110*, 2022.

[32] J. Liu, C. S. Xia, Y. Wang, and L. Zhang. Is your code generated by chatgpt really correct? rigorous evaluation of large language models for code generation. *arXiv preprint arXiv:2305.01210*, 2023.

[33] T. W. Malone and M. S. Bernstein. *Handbook of collective intelligence*. MIT press, 2022.

[34] M. V. Mäntylä, D. Graziotin, and M. Kuutila. The evolution of sentiment analysis—a review of research topics, venues, and top cited papers. *Computer Science Review*, 27:16–32, 2018.

[35] R. Mitkov. *The Oxford handbook of computational linguistics*. Oxford University Press, 2022.

[36] S. R. Moghaddam and C. J. Honey. Boosting theory-of-mind performance in large language models via prompting. *arXiv preprint arXiv:2304.11490*, 2023.

[37] A. B. Nassif, I. Shahin, I. Attili, M. Azzeh, and K. Shaalan. Speech recognition using deep neural networks: A systematic review. *IEEE access*, 7:19143–19165, 2019.

[38] J. Nielsen. How to conduct a heuristic evaluation. *Nielsen Norman Group*, 1(1):8, 1995.

[39] OpenAI. Gpt-4 technical report. *arXiv*, 2023.

[40] A. Patel, S. Bhattamishra, and N. Goyal. Are nlp models really able to solve simple math word problems? *arXiv preprint arXiv:2103.07191*, 2021.

[41] O. Press, M. Zhang, S. Min, L. Schmidt, N. A. Smith, and M. Lewis. Measuring and narrowing the compositionality gap in language models. *arXiv preprint arXiv:2210.03350*, 2022.

[42] C. Qin, A. Zhang, Z. Zhang, J. Chen, M. Yasunaga, and D. Yang. Is chatgpt a general-purpose natural language processing task solver? *arXiv preprint arXiv:2302.06476*, 2023.

[43] J. W. Rae, S. Borgeaud, T. Cai, K. Millican, J. Hoffmann, F. Song, J. Aslanides, S. Henderson, R. Ring, S. Young, et al. Scaling language models: Methods, analysis & insights from training gopher. *arXiv preprint arXiv:2112.11446*, 2021.

[44] P. Rajpurkar, J. Zhang, K. Lopyrev, and P. Liang. Squad: 100,000+ questions for machine comprehension of text. *arXiv preprint arXiv:1606.05250*, 2016.

[45] K. Reinecke and K. Z. Gajos. Labinthewild: Conducting large-scale online experiments with uncompensated samples. In *Proceedings of the 18th ACM conference on computer supported cooperative work & social computing*, pages 1364–1378, 2015.

[46] S. Ruder. Challenges and Opportunities in NLP Benchmarking. `http://ruder.io/nlp-benchmarking`, 2021.

[47] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, et al. Imagenet large scale visual recognition challenge. *International journal of computer vision*, 115:211–252, 2015.

[48] M. Sap, R. Le Bras, E. Allaway, C. Bhagavatula, N. Lourie, H. Rashkin, B. Roof, N. A. Smith, and Y. Choi. Atomic: An atlas of machine commonsense for if-then reasoning. In *Proceedings of the AAAI conference on artificial intelligence*, volume 33, pages 3027–3035, 2019.

[49] M. Sap, R. LeBras, D. Fried, and Y. Choi. Neural theory-of-mind? on the limits of social intelligence in large lms. *arXiv preprint arXiv:2210.13312*, 2022.

[50] Z. Shao, Y. Gong, Y. Shen, M. Huang, N. Duan, and W. Chen. Synthetic prompting: Generating chain-of-thought demonstrations for large language models. *arXiv preprint arXiv:2302.00618*, 2023.

[51] R. Simpson, K. R. Page, and D. De Roure. Zooniverse: observing the world's largest citizen science platform. In *Proceedings of the 23rd international conference on world wide web*, pages 1049–1054, 2014.

[52] F. Stahlberg. Neural machine translation: A review. *Journal of Artificial Intelligence Research*, 69:343–418, 2020.

[53] O. Tafjord, M. Gardner, K. Lin, and P. Clark. Quartz: An open-domain dataset of qualitative relationship questions. *arXiv preprint arXiv:1909.03553*, 2019.

[54] R. Taylor, M. Kardas, G. Cucurull, T. Scialom, A. Hartshorn, E. Saravia, A. Poulton, V. Kerkez, and R. Stojnic. Galactica: A large language model for science. *arXiv preprint arXiv:2211.09085*, 2022.

[55] M. Turpin, J. Michael, E. Perez, and S. R. Bowman. Language models don't always say what they think: Unfaithful explanations in chain-of-thought prompting. *arXiv preprint arXiv:2305.04388*, 2023.

[56] L. Von Ahn. Games with a purpose. *Computer*, 39(6):92–94, 2006.

[57] L. Von Ahn and L. Dabbish. Labeling images with a computer game. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 319–326, 2004.

[58] L. Von Ahn, R. Liu, and M. Blum. Peekaboom: a game for locating objects in images. In *Proceedings of the SIGCHI conference on Human Factors in computing systems*, pages 55–64, 2006.

[59] L. Von Ahn, B. Maurer, C. McMillen, D. Abraham, and M. Blum. recaptcha: Human-based character recognition via web security measures. *Science*, 321(5895):1465–1468, 2008.

[60] A. Wang, Y. Pruksachatkun, N. Nangia, A. Singh, J. Michael, F. Hill, O. Levy, and S. Bowman. Superglue: A stickier benchmark for general-purpose language understanding systems. *Advances in neural information processing systems*, 32, 2019.

[61] A. Wang, A. Singh, J. Michael, F. Hill, O. Levy, and S. R. Bowman. Glue: A multi-task benchmark and analysis platform for natural language understanding. *arXiv preprint arXiv:1804.07461*, 2018.

[62] X. Wang, J. Wei, D. Schuurmans, Q. Le, E. Chi, and D. Zhou. Rationale-augmented ensembles in language models. *arXiv preprint arXiv:2207.00747*, 2022.

[63] Y. Wang, Y. Kordi, S. Mishra, A. Liu, N. A. Smith, D. Khashabi, and H. Hajishirzi. Self-instruct: Aligning language model with self generated instructions. *arXiv preprint arXiv:2212.10560*, 2022.

[64] J. Wei, X. Wang, D. Schuurmans, M. Bosma, E. Chi, Q. Le, and D. Zhou. Chain of thought prompting elicits reasoning in large language models. *arXiv preprint arXiv:2201.11903*, 2022.

[65] A. J. Westphal, A. L. Butterworth, C. J. Snead, N. Craig, D. Anderson, S. M. Jones, D. E. Brownlee, R. Farnsworth, and M. E. Zolensky. Stardust@ home: a massively distributed public search for interstellar dust in the stardust interstellar dust collector. *Lunar and Planetary Science XXXVI, Part 21*, 2005.

[66] X. Ye and G. Durrett. The unreliability of explanations in few-shot prompting for textual reasoning. *Advances in neural information processing systems*, 2022.

[67] O. Yoran, T. Wolfson, B. Bogin, U. Katz, D. Deutch, and J. Berant. Answering questions by meta-reasoning over multiple chains of thought. *arXiv preprint arXiv:2304.13007*, 2023.

[68] E. Zelikman, Y. Wu, J. Mu, and N. Goodman. Star: Bootstrapping reasoning with reasoning. *Advances in Neural Information Processing Systems*, 35:15476–15488, 2022.

[69] L. Zhang, L. Dugan, H. Xu, and C. Callison-Burch. Exploring the curious case of code prompts. *arXiv preprint arXiv:2304.13250*, 2023.

[70] Z. Zhang, A. Zhang, M. Li, and A. Smola. Automatic chain of thought prompting in large language models. In *The Eleventh International Conference on Learning Representations (ICLR 2023)*, 2023.

[71] W. Zhong, R. Cui, Y. Guo, Y. Liang, S. Lu, Y. Wang, A. Saied, W. Chen, and N. Duan. Agieval: A human-centric benchmark for evaluating foundation models. *arXiv preprint arXiv:2304.06364*, 2023.

[72] S. Zhou, M. Gordon, R. Krishna, A. Narcomey, L. F. Fei-Fei, and M. Bernstein. Hype: A benchmark for human eye perceptual evaluation of generative models. *Advances in neural information processing systems*, 32, 2019.

A Scaling up user study

To see if our results generalize, we expand our user study to include 12 more participants, bringing us to a total of 32. The trends described in our main paper persist, with additional dimensions now statistical significant. In addition to the results already described in the main paper, Table 4 shows that prior language model and prompting experience significantly improve participants' performance on our problems **across all domains**. This new statistic adds further support to a point we made in our main paper: we noted that users self-reported a learning effect, where they improve their ability to write instructions as they tackle more problems.

The change in PCA clusters over time between the user study results reported in the main paper and the scaled-up user study results is depicted in Figure 3. Overall, cluster arrangements are similar across problems; however, human-generated instructions increasingly span across the first principal component for most problems, indicating increased diversity in human instructions. A greater number of instructions also provides a more reliable indication of each strategy's solvability.

Table 4: Pearson's correlation coefficient between participant attributes (demographic, background, and experience) and the maximum number of test cases passed and the time taken for each problem. '..' indicates trending towards significance ($p < 0.1$) and '*' denotes significance ($p < 0.05$). **Pass** is the average number of testcases passed and **Time** is the avg. time taken between first and last interaction with the problem

Domain →	**Biology**		**Physics**		**Math**		**Programming**		**General**		**Overall**	
Participant ↓	**Pass**	**Time**	**Pass**	**Time**	**Pass**	**Time**	**Pass**	**Time**	**Pass**	**Time**	**Pass**	**Time**
Demographics												
Age	-0.36*	-0.09	-0.57*	-0.11	-0.27	-0.04	-0.38*	0.08	-0.16	0.15	-0.44*	-0.02
Experience												
Biology	0.13	-0.06	-0.04	0.35*	0.02	-0.33..	-0.09	0.18	0.11	0.03	0.03	0.03
Physics	-0.26	-0.08	-0.01	0.37*	0.03	-0.10	-0.05	-0.14	0.06	-0.23	-0.05	-0.03
Math	-0.03	0.14	0.08	0.28	0.24	0.20	0.18	0.26	0.23	0.06	0.19	0.30..
Trivia	-0.01	-0.11	-0.00	0.14	0.37*	-0.20	0.30..	0.24	0.19	-0.12	0.22	-0.06
Experience												
LM	0.44*	0.48*	0.22	0.28	0.54*	0.37*	0.49*	0.16	0.55*	0.39*	0.58*	0.60*
Prompting	0.40*	0.34..	0.20	0.45*	0.38*	0.39*	0.35*	0.00	0.39*	0.33..	0.44*	0.58*
Programming	-0.06	0.23	0.09	0.08	0.26	0.13	0.29	0.14	0.36*	0.09	0.26	0.23

B Participant demographics

Figures 4, 5 and 6 describe the participant demographics and experience as surveyed before the study, and their feedback after they finished the study.

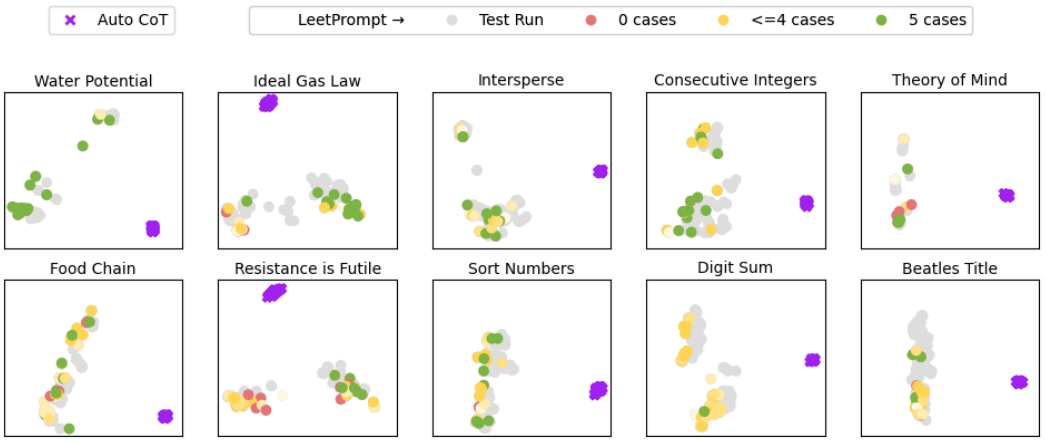
C Evaluating other models

In this section, we evaluate how the human-generated instructions work across other LLMs. All the instructions were generated using GPT-4 interactions. Here, we test if those same instructions work on GPT-3 and GPT-3.5. We also add 10 new internal test cases along with the 5 externally generated test cases reported in the main paper.

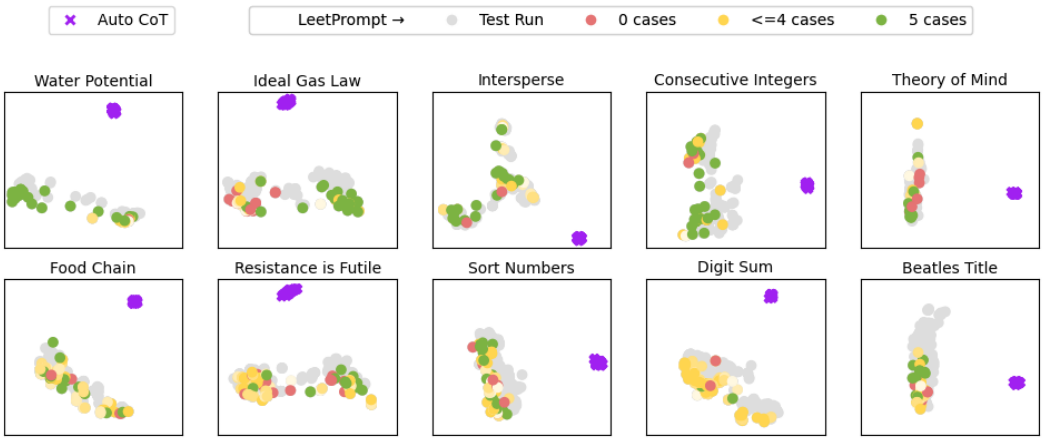
Table 7 shows the performance of the instructions using GPT-3 (text-davinci-003) as the language model. We also show the results on all test cases (5 external + 10 internal). The instructions submitted by the study participants passed 143 out of 150 test cases which surpasses all the automatic strategies. The best performing automatic method, 4-shot, passes only 86 test cases, which accounts for only 58% of the test cases that human instructions succeed on.

Table 6 shows the performance of the instructions using GPT-3.5 as the language model. The instructions submitted by the study participants passed 148 out of 150 test cases which surpasses all the baseline prompting strategies by a significant margin. Again, the best performing automatic method, 4-shot, passes only 94 test cases, which accounts for only 63% of the test cases that human instructions succeed on.

Finally, table 5 shows the performance of the instructions using GPT-4 as the language model. The number of test cases passed is higher for all instruction strategies when using GPT-3 and GPT-3.5. The instructions submitted by the participants pass all 150 test cases, while 4-shot prompting passes 125 test cases which is the highest amongst the 3 models. Therefore, automatic methods using GPT-4 only pass 83% of all the successful human generated instructions.



Collected instructions with 20 participants (dated **May 17th**: NeurIPS 2023 paper submission deadline)



Collected instructions with 32 participants (dated **May 24th**: NeurIPS 2023 supplementary deadline)

Figure 3: **Visualizing the space of Auto-CoT and LEETPROMPT Instructions**: 2D Principal Component Analysis (PCA) of embeddings of Auto-CoT and collected instructions from LEETPROMPT. Auto-CoT instructions are marked as purple X. LEETPROMPT instructions are marked as dots with colors representing its solvability: "Test" instructions are colored gray. For "Submitted" instructions, red color indicates that they failed all testcases, yellow indicates they passed 1-4 testcases, and green indicates that they passed 5 testcases. Instructions are specifically shown for two problems each from each domains to illustrate the different strategies used by participants and whether they were successful.

Overall, we can see that despite the model we use, the human-generated instructions consistently outperform the automatic strategies. Even on less powerful models like GPT-3 and GPT-3.5 the human instructions pass more than 95% of the test cases, demonstrating the importance of studying LLM capabilities with human interactions.

D Diversity of problems

We provide (in Table 8) a summary of all 10 problems with the description of each problem and some example input-output pairs that participants were shown as part of the problem description. Below, we list the reasons for choosing each of these problems to include in our user study.

Water potential. We chose “Water potential" because it is a very simple problem in biology. We wanted to mix simple and difficult problems to see how the task’s complexity influenced how users developed instructions. Most users noticed the greater than/less than relationship with 10, but even if

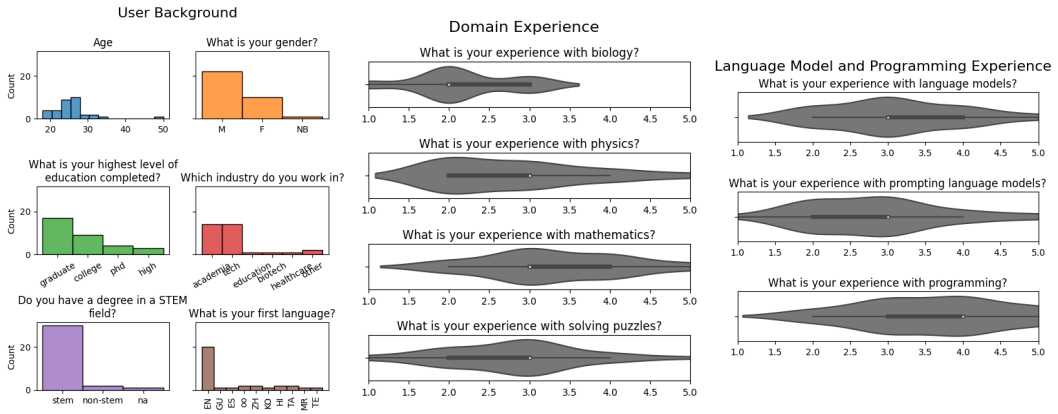


Figure 4: **Pre-study survey of user study participants** indicating their background (age, gender, highest level of education, industry, type of degree, and their first language), experience with different subjects (biology, physics, mathematics, and puzzles), and experience with using information technologies relevant to our study (language models, prompting, and programming)

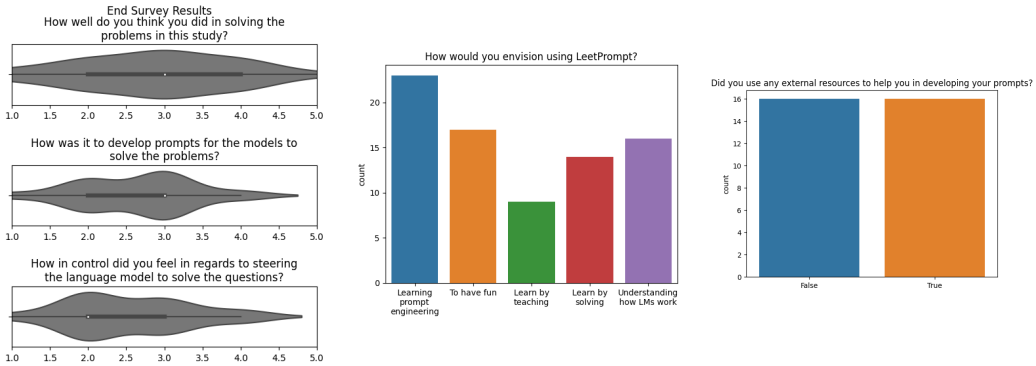


Figure 5: **Post-study survey of user study participants** describing their experience with solving problems, their perception of the platform and feeling of control with language models. Participants also report on whether they used external resources while solving problems and how they envision using leetprompt platform in the future.

they didn't, copying and pasting the problem statement and providing a few examples worked fairly well.

Food chain. "Food chain" is a more difficult biology problem that the problem setters were unable to solve. It required much more complex logic that differed depending on the relative position of the two species in the food chain. Some participants asked the model how to solve it and gave that back to the model, which worked fairly well, while others gave incorrect logic, which worked in a few cases despite being factually incorrect.

Ideal gas law. We chose "Ideal gas law" because it is one of the most fundamental equations in physics. Users didn't have to do much except apply the equation, which they could easily reduce to a simple division by 100, as many quickly realized. With this problem, however, a copy-paste strategy or even leaving out the explanation and asking the model to detect a pattern worked extremely well.

Resistance is futile. "Resistance is futile" necessitated more logic in calculating the total resistance of the circuit prior to applying Ohm's Law. Despite the fact that Ohm's Law is fairly simple, the equations for calculating total resistance were too complex for the language model, and some participants found the text description of the circuit difficult to interpret. No one was able to solve this without simplifying the formula in our first round of user studies, but one participant in the second round was able to solve it with a vague prompt that did not include a formula and some examples.

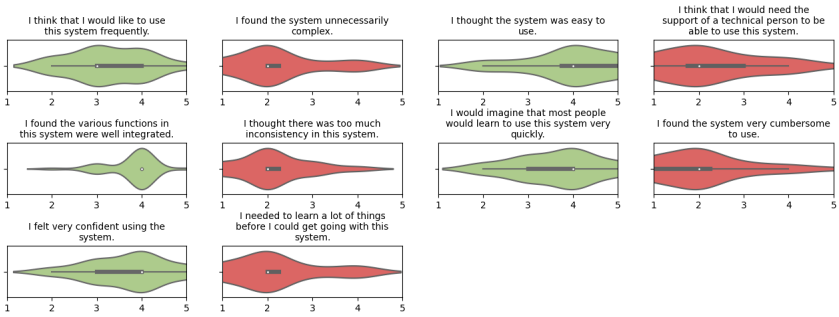


Figure 6: **System Usability Scale** [4] used for measuring the usability of LEETPROMPT by study participants.

Consecutive integers. "Consecutive integers" is the simpler of the two math problems. It is an elementary school level problem that the language model easily understands. Users who simplified the formula were successful, but it was also possible to solve the problem by simply pasting or rewording the prompt and providing examples.

Digit sum. "Digit sum" is an intriguing problem because it is very simple for a human to solve and is also considered an elementary school level problem. However, the logic is much more difficult to explain to the language model because the model isn't as strong in math and, in many cases, doesn't know what a "digit" is. Participants were surprised by the resulting outputs of their test inputs, and found it difficult to understand why the model produced those results. Even though it was very simple to solve manually, the problem setting team was unable to solve it using the language model. In this problem, only two instructions worked; both used a rewording of the question and two examples that were the same and in the same order, and neither example was an edge case example.

Intersperse. "Intersperse" is a simple programming problem that participants with limited programming experience could understand. The problem setting team derived this problem from an open dataset [8] rather than creating it. Some participants were surprised by the output because it provided a code to solve the problem rather than the solution, with one participant even adding "Please no code" to their prompt.

Sort numbers. "Sort numbers" is another relatively simple programming problem, a simple array sort with English numbers rather than numerals. A version of this problem is also used in the open dataset [8]. The majority of participants explicitly converted between the text versions of the numbers and the numerals and created an array, while some successfully sorted the words directly.

Beatles. We selected the "Beatles problem" because it is more concerned with general knowledge and text processing than with mathematical formulas or programming logic. To identify Beatles songs, the model needed to recognize them and be able to parse an input string. The model had trouble recognizing the song "Rain" in one of the example problems, which stumped participants who were trying to pass every example case before submitting, but because it was not used in the test cases, those who submitted anyways passed. The language model also counted additional titles that were semantically similar to phrases in the passage but were not direct substrings.

Housesitting. "Housesitting" is a theory of mind problem. The problem-solving team wanted to know if the language model could perform well in theory-of-mind tasks. Participants were required to explain its lengthy description to the model. However, once participants provided all of the scenario descriptions, the language model was mostly successful in solving the problem. However, because of time constraints and a general dislike of reading long problem descriptions before being able to solve the problem, some participants were discouraged from even attempting the problem. The standard strategy participants employed, which was mostly successful, was to copy and paste the scenario description and insert some examples.

Comparison of existing prompting approaches to prompts collected from LEETPROMPT using GPT-4, GPT-3.5-TURBO and GPT-3 as the language models. **0s**: Zero-shot; **0s CoT**: Zero-shot Chain-of-Thought prompting; **1s, 2s, 3s, 4s**: 1,2,3,4 shot (or examples) prompting; **Ours**: Prompts from LEETPROMPT. **P** denotes the maximum number of testcases that the the given method was able to pass.

Table 5: GPT-4

Domain	Question	0s P	CoT P	1s P	2s P	3s P	4s P	Ours P
Biology	Water Potential	7	7	8	**15**	**15**	**15**	**15**
Biology	Food Chain	9	8	9	6	6	8	**15**
Physics	Ideal Gas Law	12	7	10	12	12	13	**15**
Physics	Resistance is Futile	3	0	3	10	11	10	**15**
Math	Consecutive Integers	14	14	13	14	13	13	**15**
Math	Digit Sum	6	6	9	8	8	9	**15**
Programming	Intersperse	13	13	14	14	14	14	**15**
Programming	Sort Numbers	0	0	13	13	13	14	**15**
Knowledge	Beatles Title	14	14	11	14	14	14	**15**
Knowledge	Theory of Mind	1	0	**15**	**15**	14	**15**	**15**
		79	69	105	121	120	125	**150**

Table 6: GPT-3.5-TURBO

Domain	Question	0s P	CoT P	1s P	2s P	3s P	4s P	Ours P
Biology	Water Potential	3	9	12	14	14	**15**	**15**
Biology	Food Chain	0	1	9	6	6	8	**15**
Physics	Ideal Gas Law	0	3	4	10	13	11	**15**
Physics	Resistance is Futile	0	0	0	2	2	2	**15**
Math	Consecutive Integers	10	13	14	11	12	8	**15**
Math	Digit Sum	7	3	5	6	6	5	**14**
Programming	Intersperse	6	2	1	11	11	12	**15**
Programming	Sort Numbers	1	0	11	12	12	13	**15**
Knowledge	Beatles Title	11	10	11	12	12	11	**15**
Knowledge	Theory of Mind	2	1	10	8	7	8	**14**
		43	33	73	88	92	94	**148**

Table 7: GPT-3

Domain	Question	0s P	CoT P	1s P	2s P	3s P	4s P	Ours P
Biology	Water Potential	6	0	8	10	12	14	**15**
Biology	Food Chain	9	8	8	10	9	11	**15**
Physics	Ideal Gas Law	0	0	6	7	8	8	**15**
Physics	Resistance is Futile	0	0	0	3	4	4	**15**
Math	Consecutive Integers	7	7	9	4	4	6	**15**
Math	Digit Sum	4	3	4	3	3	5	**12**
Programming	Intersperse	0	0	4	9	10	10	**13**
Programming	Sort Numbers	0	0	6	5	2	9	**15**
Knowledge	Beatles Title	9	11	10	9	7	8	**15**
Knowledge	Theory of Mind	0	0	9	10	10	10	**13**
		32	38	68	74	71	86	**143**

E Qualitative coding

To gauge the diversity in responses, we implemented qualitative coding. This method is typically used in social sciences to categorize and analyze qualitative data - in this case, submissions made in response to instructions. In this process, we assign codes, or specific labels, to different aspects of the data in order to classify it in a meaningful way.

Here are the codes that we utilize:

Instruction prompting (`INST`): This coding category pertains to strategies involving direct instructions. These are the most common methods employed by participants. This might involve:

- `INST-SIM`: Simplifying the problem to make it more understandable.
- `INST-EXP`: Asking the model to emulate a third party, like an expert or a crowd, to generate a response. [1]

Table 8: **Summary of Problems** given in the user study, with an input / output example.

	Question	Description	Example
Biology	Water potential	*Given the sucrose concentration of an animal cell in a solution, determine whether it will shrink or expand*	`INPUT: 11` `OUTPUT: expand`
	Food chain	*Given a food chain, determine whether an increase in one given species will lead to an increase or decrease in the population of a second species.*	`INPUT: kelp -> sea urchin -> otter -> orca, otter, kelp` `OUTPUT: increase`
Physics	Ideal gas law	*Deriving final pressure with constant volume using the ideal gas law PV = NRT*	`INPUT: 400` `OUTPUT: 4`
	Resistance is futile	*Determining current for an electrical circuit given a voltage and resistance.*	`INPUT: 100` `OUTPUT: 75`
Math	Consecutive integers	*Given a sum of three consecutive integers, find the smallest integer.*	`INPUT: 63` `OUTPUT: 20`
	Digit sum	*Given two input numbers, output the smallest number with the given number of digits and given digit sum.*	`INPUT: 4, 9` `OUTPUT: 1008`
Programming	Intersperse	*Insert a number 'delimeter' between every two consecutive elements of input list 'numbers'*	`INPUT: [1, 2, 3], 4` `OUTPUT: [1, 4, 2, 4, 3]`
	Sort numbers	*Given an input of space-delimited string of numberals from 'zero' to 'nine', sort them from smallest to largest*	`INPUT: 'three one five'` `OUTPUT: 'one three five'`
General Knowledge	Beatles	*Given a funny phrase, how many Beatles song titles are in it?*	`INPUT: Yesterday I toured a yellow submarine.` `OUTPUT: 2`
	Housesitting	*Theory-of-mind: Bob has to go on a trip for his job. He has to leave his house - and his dog - for a week while he's on the trip. He is having his friend Anna take care of the house and the dog, Fido, while he's away.... Anna completes the given action. When Bob comes back, where will he look for the given item? Will he find it?*	`INPUT: Anna takes out Fido's treat bag to feed him after he sits on command. She leaves the bag on the counter. Bob comes back and Fido welcomes him like a very good boy! Bob wants to feed him a treat.` `OUTPUT: G, no`

- `INST-INC`: Includes instructions that may not be factually correct but can still assist the language model in problem-solving. [55]

Examples (`EX`)**:** This category involves providing examples, which have been observed to enhance the model's problem-solving capacity.

- `EX-ZERO`: No examples are provided, leaving the model to interpret potential inputs and outputs. [26]
- `EX-N-SHOT`: Few examples are given, providing some clues to the model.
- `ORDER`: The solution utilized an unusual sequence in which examples are presented. [62]

Chain of thought (`COT`)**:** Encouraging the model to break down the explanation into steps or providing step-by-step problem-solving instructions can enhance the model's performance.

- `COT-CONS`: Changing the decoding strategy to promote diverse sampling with self-consistency. [63]
- `COT-COMP`: Using complex reasoning steps to assist the model. [15]
- `COT-TEXT`: Indicates that a chain of thought approach doesn't substantially help with text-based problems. [66]

Structure (`ST`) The way the prompt text is formatted can influence the model's performance.

- `ST-NONE`: Continuously formatted instructions without line breaks. These appeared to be less effective.
- `ST-BREAK`: Breaking the prompt into multiple lines. The most commonly employed formatting strategy.
- `ST-STEP`: Structuring the prompt with steps or bullet points. This was shown to enhance prompt clarity.
- `ST-QUES`: Using "Q:" instead of "Question:". We observed this to be effective in certain cases. [15]

Writing code (`CODE`): Writing code (CODE): This category pertains to responses involving coding. [69]

- `CODE-PSEU`: Writing solutions in pseudocode format.
- `CODE-PROG`: Writing actual functions in a programming language or asking the model to generate code.

Asking the model for help (`SELF`): In some instances, even if the model doesn't have a final solution, it can still provide helpful input as an intermediate step toward helping the user create a solution.

- `SELF-ASK`: Repeatedly asking the model for help can yield beneficial results. [41] (SELF-ASK)
- `SELF-TAUG`: When prompted with a few rationale examples as a self-taught-reasoner, the model generates rationales to answer many questions. [68]

Strategies for future studies We found more strategies that we didn't code for in our responses, but we expect to appear in future studies with the platform.

- Social engineering the model such as giving it confidence or threatening it [13]. We did not see any of our participants do this, but they may attempt in future studies, as more awareness of this technique percolates to the public.
- Generate programs as the intermediate reasoning steps, but offloads the solution step to a runtime such as a Python interpreter [18]. We did not have any integrations with any code interpreters, but if we were to build such a feature into LEETPROMPT this would be useful coding.
- 'Program of Thoughts' (PoT) uses language models to express the reasoning process as a program and executes the code on an external computer. [9]. We did not have any integrations that run code, but if we were to build this feature into LEETPROMPT, this would be a useful coding.

F Diversity of instructions

In this section, we visualize some example human-generated instructions from our user study, how many test cases the instruction passed, and how we codified the instruction strategy with an explanation for our code. The following are a few sample instructions submitted for the problem "Resistance is Futile":

Human-generated instruction:

```
Find the value of 7500 / [[ INPUT ]].
Give your answer in the form "Output: "
```

Number of test cases passed: 5/5

Explanation: This solution presents a direct instruction approach, incorporating a simplified formula without providing any examples. Given its simplicity, there's no break line, list or question/answer structuring within the prompt itself.

Coding: INST-SIM, EX-ZERO, ST-NONE

Human-generated instruction:

```
For the following resistance, output the total current of the circuit
Input: 100
Output: 75
Input: 250
Output: 30
Input: 300
Output: 25
Input: [[ INPUT ]]
```

Number of test cases passed: 5/5

Explanation: This solution uses direct instruction, void of any simplified formulas. It reconstructs the question into a more straightforward form, stripping away details that might seem essential to a human, such as the configuration of the resistors or the battery's voltage. Nonetheless, the model manages to infer an answer from the three examples provided, which are separated by new lines. Interestingly, the sequence of these examples deviates from the original list.

Coding: INST, EX-3-SHOT, ST-BREAK, ORDER

Human-generated instruction:

```
Here is a electrical circuit question I am trying to solve:
What is the total current in milliamperes of the circuit?

Based on ohm's law, current is voltage devide by resistance,
the circuit has a voltage source of 9 volts, I will provide a number,
the resistance of whole circuit will be 1.2 times the number I provided,
can you calcuate the current for me?

The following are some example inputs and outputs:
Input:  100
Output:  75
Input:  300
Output:  25
Input:  750
Output:  10

Please note that the output of the current is milliampere, just output
a number without unit

Input:  [[ INPUT ]]
```

Number of test cases passed: 5/5

Explanation: This solution employs a 'chain of thought' strategy, as it involves the participant explaining the calculation process for the current using a somewhat simplified formula. It also demonstrates the participant's effort to correct the model's behavior by explicitly stating that the output should be in milliamperes at the conclusion of the prompt.

Coding: INST, COT, EX-3-SHOT

Human-generated instruction:

```
Tell language model how to solve the problem here
You are building a device to resist the Borg.  In order to do this, you
need to connect some resistors in a circuit with a 9 V battery.  You have
5 resistors of a given resistance (in ohms).  You plan to connect 3 of
them in series in parallel with two of them in series.  For the three
resistors in series, we simply add their resistances:
Rseries = R1 + R2 + R3.  For the two resistors in series, we add their
resistances:  Rparallel = R4 + R5.  Then we can calculate the equivalent
resistance of the two sets of resistors in parallel:  1/Rparallel =
1/Rseries + 1/Rparallel.Finally, we can use Ohm's Law to calculate the
total current in the circuit:  I = V/Rtotal.  What is the total current in
milliamperes of the circuit?

Input:  100
Output:  75
Input:  300
Output:  25
Input:  750
Output:  10
Input:  250
Output:  30
Input:  [[ INPUT ]]
```

Number of test cases passed: 4/5

Explanation: In this submission, the participant restates the problem statement while incorporating additional instructions and logical steps needed to calculate the current, a strategy characteristic of a 'chain of thought' prompt. The participant also provides four example cases, adhering to the order in which they were presented on the platform.

Coding: INST, COT, EX-4-SHOT, ST-BREAK

Human-generated instruction:

```
def total_current(resistance, battery_voltage):
    # Calculate the total resistance of the circuit
    series_resistance = 3 * resistance
    parallel_resistance = resistance + resistance
    total_resistance = series_resistance + (1 / parallel_resistance)

    # Calculate the total current in milliamperes using Ohm's law
    total_current = battery_voltage / total_resistance
    total_current_milliamps = total_current * 1000

    return total_current_milliamps

# Example usage
resistance = 100 # ohms
battery_voltage = 9 # volts
total_current_milliamps = total_current(resistance, battery_voltage)
print(""The total current in milliamperes is:"", total_current_milliamps)

Input: Any number
Output: total_current_milliamps
Input: [[ INPUT ]]
```

Number of test cases passed: 0/5

Explanation: This submission used Python code to create a solution that calculated the total current in milliamperes of the circuit. This required a detailed explanation of the current calculation method, which suggests the use of a 'chain of thought' prompt strategy. However, the method used was

incorrect. Direct instructions were also a feature of this approach, as they guided the model to print a specific statement. Interestingly, this participant did not provide any example input-output pairs. The structure of the solution was enhanced by placing instructions on separate lines.

Coding: `INST-INC`, `COT`, `CODE-PROG`, `EX-ZERO`, `ST-BREAK`

Human-generated instruction:

```
You are an expert electrictian.
I give you 5 resistors all of the same resistance as input.  In your
circuit is a 9 V battery.
You have connected 3 resistors in series, which is in parallel with
2 other resistors that are in series.  What is the total current in
milliamperes of the circuit?

Examples:
Input:  100
Output:  75

Input:  300
Output:  25

Now it's your turn:
Input:  [[ INPUT ]]
```

Number of test cases passed: 3/5

Explanation: This participant has asked the model to simulate an expert electrician when giving instructions. They have also provided two examples which are separated by new lines.

Coding: `INST-EXP`, `EX-2-SHOT`, `ST-BREAK`

Participants demonstrated a diverse range of strategies in attempting to solve the problem, illustrating the rich array of thought processes that emerges when different individuals tackle the same challenge. However, the effectiveness of these strategies varied. Writing code and instructing the model to impersonate an expert was less successful for this problem. Alternatively, strategies that simplified the problem were more effective. This was seen in both the transformation of the problem into a straightforward formula, and the removal of seemingly crucial problem parameters. It turns out that, in many cases, these elements were important from a human perspective but not necessary for the language model to infer a solution. In conclusion, strategies that focused on distilling the problem to its core components were typically more successful.

G User interface design

The platform underwent multiple iterations to improve user experience and testing.

Initial design. Overall, the initial UI lacked clarity in presenting primary actions and information. We brought in UI/UX engineers onto our team to improve the design. With them, we identified several areas for improvement. We describe these improvements in Figure 7

Design iterations. Our team iterated through different versions in order to address the areas identified above. In Figures 8, 9, you can see two different designs with features used in the final user study design.

Final UI design. The final user study incorporated elements from previous iterations (see Figures 11, 12, 13). To improve usability, we included a tour of the interface (Figure 10), a problem navigation pane(Figure 11), and a test feedback UI change (Figure 14).

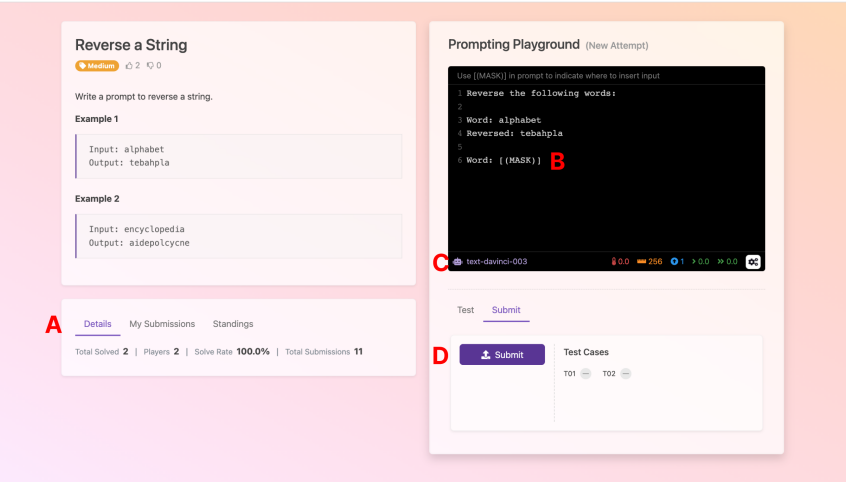


Figure 7: **Initial design details.** (A) The area containing "Details," "My Submissions," and "Standings" was located underneath problem description. If the problem description is too long, user may not see this information. (B) "Mask" represented the input that users could manipulate in order to test instructions, but the term was confusing to many users. (C) Unclear that user can switch to different models or change parameters. Icons do not clearly indicate what types of parameters can be changed. (D) Buttons to test and submit instructions were located under respective tabs. Users found this frustrating as testing and submission required an extra click into the tab before clicking on respective button.

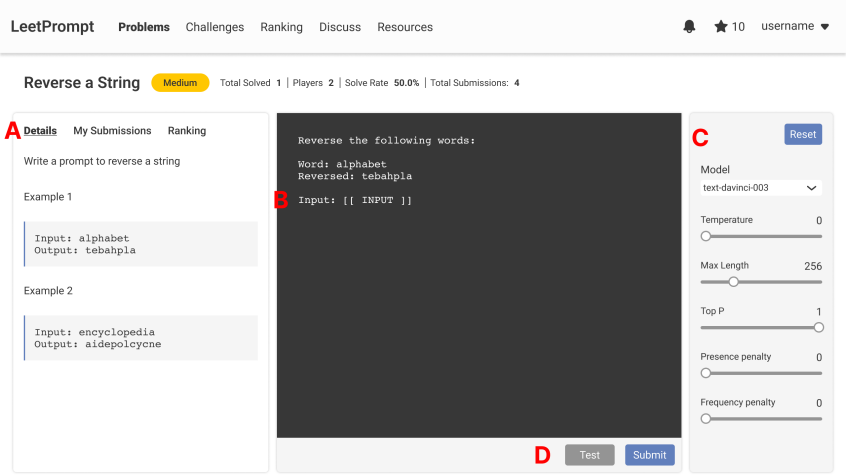


Figure 8: **Iteration 1.** (A) Problem description, details, user submissions, and user ranking were consolidated into one portion so users can easily scan for information available to them. The information previously found under "Details" was placed next to the problem name. "Details" in this iteration shows the problem description and examples. (B) The word "MASK" is replaced with the word "INPUT" so users understand that this text is to include their manipulated input. (C) Adjustment for models and parameters made more visible. (D) Test and submit buttons are visible.

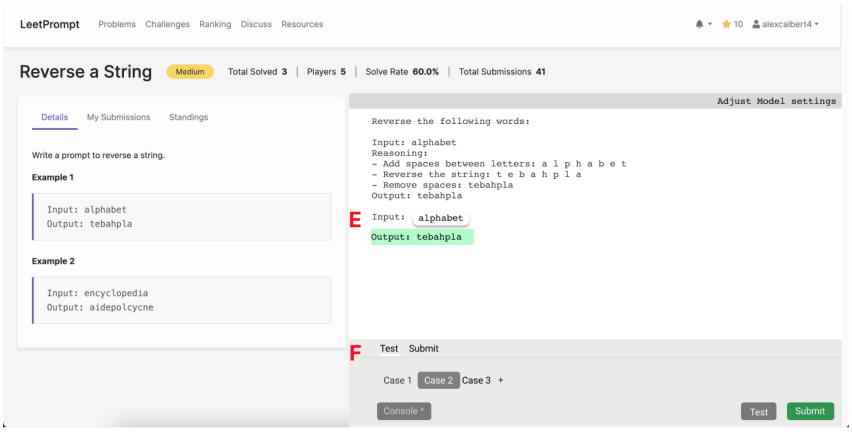


Figure 9: **Iteration 2.** In this second iteration, we explored to idea of (E) consolidating inputs and outputs into one area. Inputs would be highlighted based on the test case selected. And outputs would be highlighted in color. (F) Console expands and collapses so users have maximum area to work on instructions.

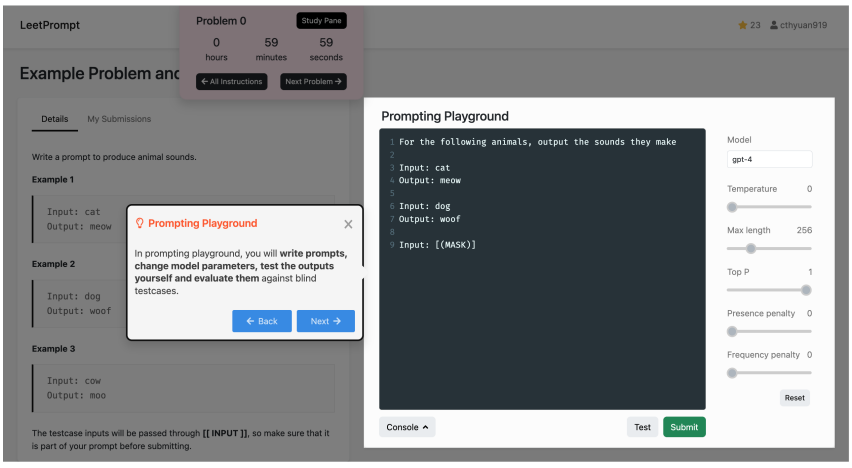


Figure 10: **Example problem and tour of the interface.** Before starting the user study, participants are given an example problem. A walk-through with tooltips introduces each section of the interface.

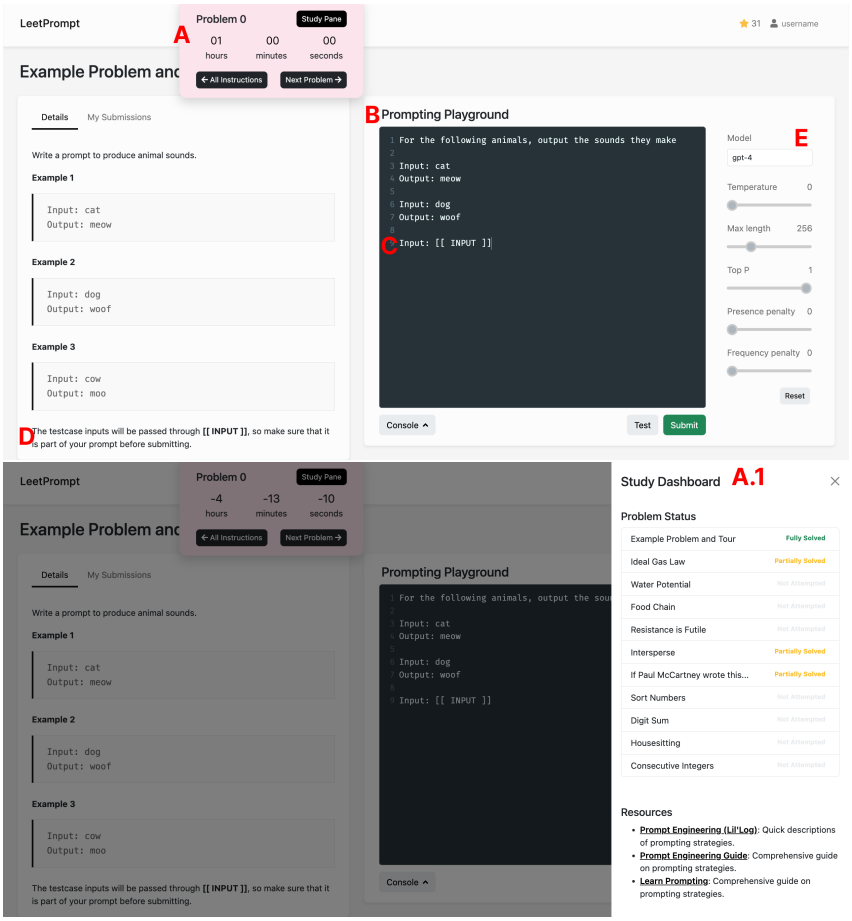


Figure 11: **User study interface.** For the user study, we adapted the interface to provide additional information to the study participants. For this, we made the following changes as marked: (A) Modal overlay provides problem navigation and shows time remaining in the study. The button "Study Pane" leads to a study dashboard side sheet (A.1) where participants can navigate to other problems and see the status of each ("Fully Solved," "Partially Solved," "Unsolved," and "Not Attempted). (B) Explicitly stating the name of this section. (C) & (D) Clarification of relationship between input and test cases. (E) Model and parameters adjustments are visible but frozen for the purposes of this study. This is explained in the initial walk-through.

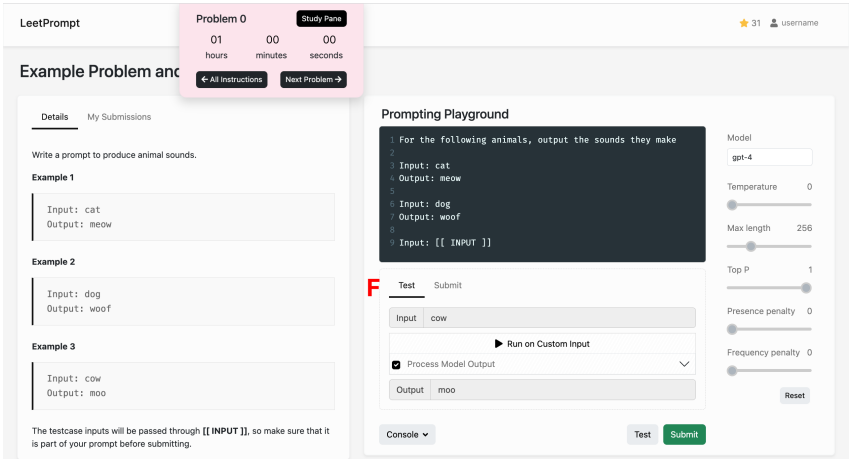


Figure 12: **Testing functionality.** The participants can test their instructions with their own custom inputs in the test console. The specific functionalities are as follows: (F) Participant's text input was divided into three sections. The top screen allows editing for instructions. Bottom section shows area to enter and edit input. Area that shows output of model is below. Participant can either click on "Run on Custom Input" or the "Test" button to test instructions on a particular input.

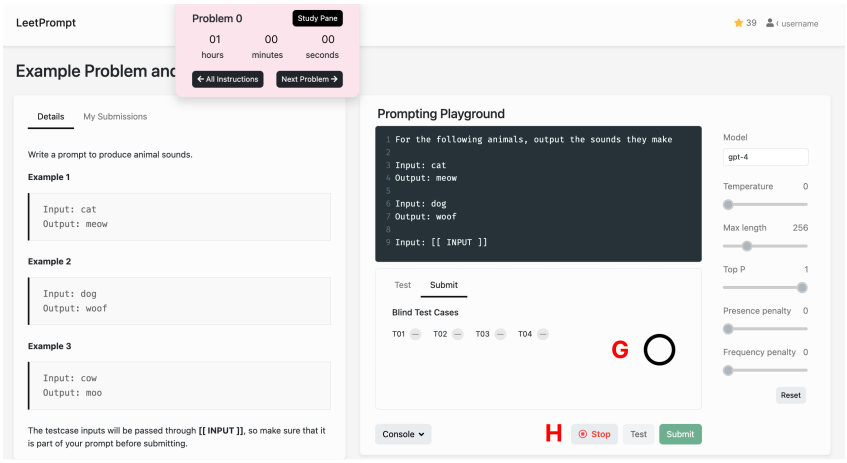


Figure 13: **Submit functionality.** After the participants are done testing their instructions, they can submit their prompt for evaluation against the blind testcases. When the "Submit" button is clicked, participants are taken to the submit tab. (G) Loading animation was added to indicate progress. (H) Stop button added for better user control.

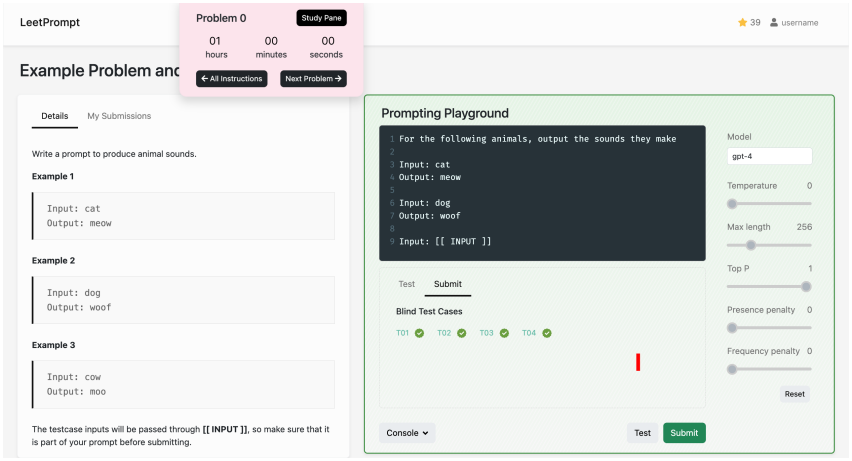


Figure 14: **Submission feedback** Once a submission is completed, participants receive feedback on how their instructions performed against the blind test cases. The feedback is shown to the participants in form of the shown the number of test cases that passed (see I). Instructions are highlighted based on number of test cases passed. Green indicates that all test cases passed, yellow for some test cases passed, and red for no test cases passed.