What You Say = What You Want?
Teaching Humans to Articulate Requirements for LLMs

Qianou Ma
qianouma@cmu.edu
Carnegie Mellon University
Pittsburgh, PA, USA

Weirui Peng
wp2297@columbia.edu
Columbia University
New York, NY, USA

Hua Shen
huashen@umich.edu
University of Michigan
Ann Arbor, MI, USA

Kenneth Koedinger
koedinger@cmu.edu
Carnegie Mellon University
Pittsburgh, PA, USA

Tongshuang Wu
sherryw@cs.cmu.edu
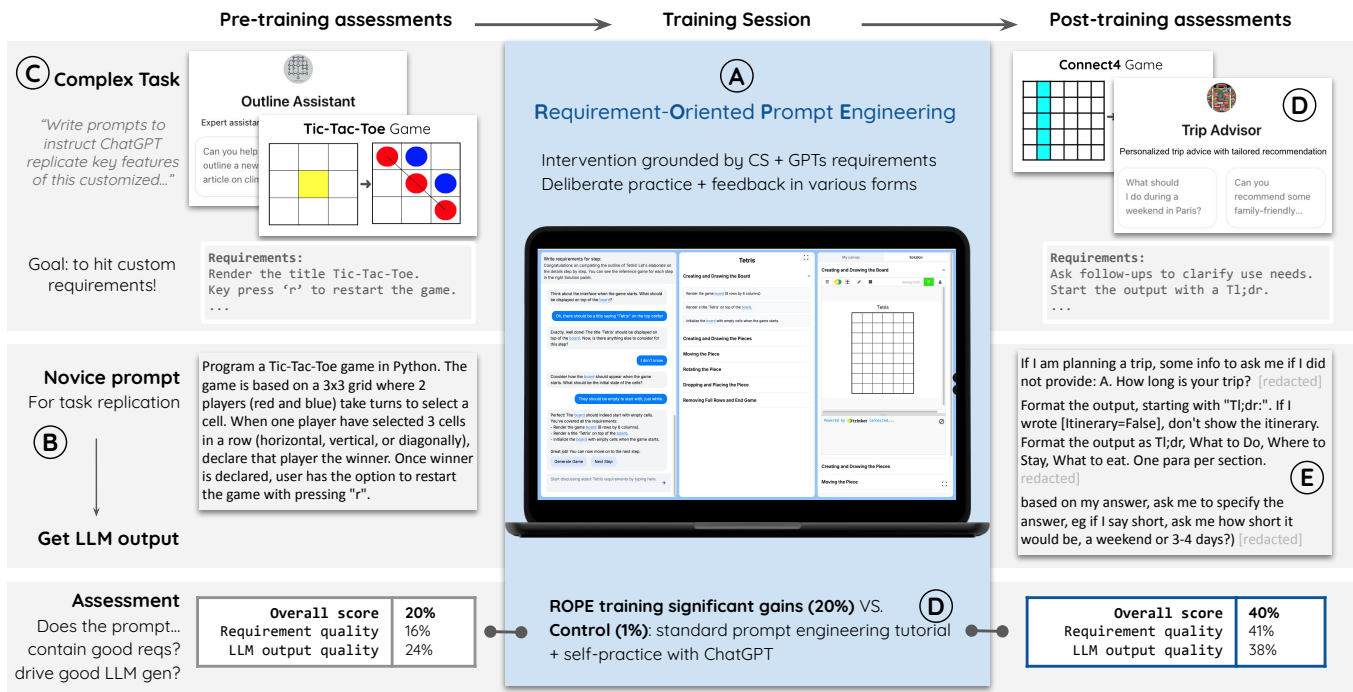Carnegie Mellon University
Pittsburgh, PA, USA

Figure 1: We propose Requirement-Oriented-Prompt Engineering (ROPE) to help novices write effective prompt programs by focusing on articulating good requirements. (A) Our training provides deliberate practice in refining requirements, with automated feedback across various modalities. (B) We provide assessments that link requirement quality to LLM output and evaluate on these assessments, in a counter-balanced pre-post experimental design. (C) Through pre-post assessments where novices write prompts to replicate existing programs (e.g. replicating Trip Advisor in D through a prompt in E), we observe that ROPE significantly improves novices' prompts and requirements compared to traditional prompt engineering training.

ABSTRACT

Prompting ChatGPT to achieve complex goals (e.g., creating a customer support chatbot) often demands meticulous prompt engineering, including aspects like fluent writing and chain-of-thought techniques. While emerging prompt optimizers can automatically refine many of these aspects, we argue that clearly conveying customized requirements (e.g., how to handle diverse inputs) remains a human-centric challenge. In this work, we introduce Requirement-Oriented Prompt Engineering (ROPE), a paradigm that focuses human attention on generating clear, complete requirements during prompting.

We implement ROPE through an assessment and training suite that provides deliberate practice with LLM-generated feedback. In a study with 30 novices, we show that requirement-focused training doubles novices' prompting performance, significantly outperforming conventional prompt engineering training and prompt optimization. We also demonstrate that high-quality LLM outputs are directly tied to the quality of input requirements. Our work paves the way for more effective task delegation in human-LLM collaborative prompting.

CCS CONCEPTS

• **Human-centered computing** → **Interactive systems and tools**; • **Applied computing** → **Computer-assisted instruction**; • **Computing methodologies** → Natural language generation.

KEYWORDS

LLM, prompt engineering, requirement engineering, end-user programming

1 INTRODUCTION

General-purpose AIs like large-language models (LLMs) have evolved from simple next-word predictors [7] to powerful assistants capable of fulfilling complex user needs [23, 48, 53]. This improvement in LLM instruction-following has encouraged users to delegate increasingly intricate tasks to these models. In the early days of prompt engineering, users primarily focused on refining the wording of simple instructions to improve LLM output quality [26, 61]. Today, prompts resemble detailed "essays" that define LLM roles, human preferences, and other task-specific requirements. Rather than one-off small requests, these prompts start to power the generation of *programs*. On one hand, designers and developers now write functional descriptions for LLM agents (e.g., Devin, SWE-Agent) to translate into executable software code [27, 58, 68, 72]. On the other hand, *everyday users* can write complex prompts to tailor general-purpose LLMs into special purpose LLM Applications. For instance, an LLM app (or a GPTs) like `Trip Advisor`[1] (Figure 1D) can be built solely using prompts similar to Figure 1E. These LLM applications function similarly to traditional software and are accessible through LLM App Stores like GPT Store [1] and FlowGPT [20], where they can be reused with a single click. In just a few months, over 30,000 "LLM app developers" have created millions of GPT apps, with the most popular ones being used more than 5 million times [63, 78]. This trend signals a future of **end-user programming through natural language**, where anyone can create reusable programs just by writing prompts.

The future of writing prompt programs looks promising, but *status quo* prompt engineering remains a challenge. Various studies show that optimizing LLM outputs requires well-rounded prompts, involving fluent writing, clear instructions, personas, formats, and

effective adaption of prompting techniques like Chain-of-Thought [8, 16, 21, 40, 41, 57]. These complexities often confuse novices, causing them to make *ad hoc* revisions to prompts without a clear understanding of what needs improvement [77]. Recognizing the challenge, the NLP community is developing *prompt optimizers* to automate some prompt refinement steps, e.g., incorporating role-plays, applying effective techniques, structuring prompts, and enhancing text fluency (e.g., Figure 2, discussions in Section 2.2).

The rise of optimizers makes us think: What aspects of prompt engineering should humans still master? We argue that the remaining part of manual prompt creation is akin to the core step before programming: **requirement articulation**, where *natural language requirements* are used to define and document how a system should behave, including expected inputs and outputs [35] (e.g., "`Ask follow-ups`" for `Trip Advisor` app in Figure 1). Requirements are the backbone of program functionalities. The prompt in Figure 2 drives a complex LLM adaptation because the user manually provided specific requirements. **Customized requirements are hard to predict, and thus difficult to fully automate and must be written by humans.** We argue that in a future where prompt programs are common, individuals should develop proficiency in requirement articulation. In other words, **we need to shift our focus towards Requirement-Oriented Prompt Engineering, a novel paradigm that we call "ROPE".**

Articulating clear and complete requirements is a known challenge in software engineering [37, 44, 46], and poor requirements often cause software failures [42]. This difficulty is amplified by the unpredictable nature of LLMs, which complicates controlling outputs [12, 15]. Even experts need multiple iterations to refine requirements [59], and novices frequently make requirement mistakes [5]. Recognizing both the importance and difficulty of this task, we pose the research question: **Can we help end-users better instruct LLMs to achieve their goals through ROPE training?**

To answer this question, we develop the ROPE paradigm (Section 3), along with the requirement-focused training and assessments to (1) help users practice writing complete and correct requirements, and (2) deepen our understanding of how requirements affect the quality of LLM outputs. Specifically, we formalize *tasks* (Section 4.1) with complex requirements (e.g., game development), and design corresponding *assessments* (Section 4.2) that can measure the requirement quality, the LLM output quality, and their correlations. We also design and implement an interactive training mechanism (Section 4.3) that *disentangles requirement articulation from other adjacent prompting aspects*. Our system supports deliberate practice on requirement refinement by generating immediate feedback grounded on oracle requirements.

We conduct a randomized controlled user study (Section 5) with 30 prompting novices, comparing the requirements users write before and after training, and against a control group that received standard prompt engineering training. We confirm that *our requirement-focused training is effective* (Section 6): novices receiving this training significantly improved their requirement performances by two-fold in the pre-to-post test, achieved significantly more gains than the control group, and learned to iterate on prompts in a more structured way towards requirements. Moreover, we confirm that *requirements are indeed important for the LLM generation*: There is a strong positive correlation between the
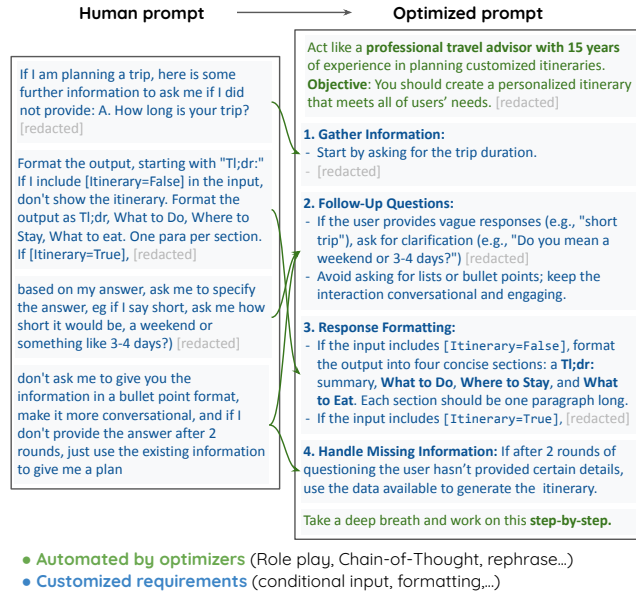
---

[1]https://chatgpt.com/g/g-K38J9feSL-trip-advisor

Figure 2: Automatically refine a user's prompt for `Trip Advisor` using the optimizer Prompt Maker. Customized requirements still need to come from the human prompt, while the optimizer improves on fluency, role plays, structures, etc.

requirement qualities and the LLM output qualities. Importantly, *the training gains are complementary to potential improvements from optimizers*: Using a popular prompt optimizer, we observe that optimizers can indeed improve requirement quality, but they cannot close the gap between the baseline and the training groups. We conclude with an in-depth discussion (Section 7) on future directions for human training and optimizer design to enhance human-LLM complementarity in ROPE. In summary, we contribute:

- We identify the significance of requirement-based prompting, and propose the ROPE paradigm as a foundation for future human-LLM task delegation in prompt engineering.
- To the best of our knowledge, we are the first to investigate training specifically for requirement-based prompting. We developed tasks, assessments, and training systems that effectively improve requirement generation skills.
- We provide both quantitative and qualitative insights for the future of prompting, highlight the relationship between the quality of requirements and LLM performance, and discuss current optimizers' limitations and potentials.

As natural language prompt programs become more widespread, everyone needs the skill to write good requirements. We believe that with our ROPE paradigm, LLMs can empower anyone to program reusable AI assistants.

2 RELATED WORKS

We review three key areas relevant to our work: prompt engineering, prompt optimization in natural language processing, and requirement engineering in end-user software engineering. We highlight a significant gap in the literature: the lack of requirement-oriented prompt engineering training for end users, which we propose to address in this paper.

2.1 Current Prompt Engineering Practices and Challenges

Prompt engineering (PE) has been essential for crafting effective input instructions to guide LLMs toward generating desired outputs [6, 38, 77]. However, the non-deterministic nature of LLMs makes it challenging for humans to predict LLMs' behavior across prompts [15, 49]. This leads to wasted time on unproductive strategies, such as trivial wording changes [15, 19, 77]. While some challenges can be mitigated through automation (see Section 2.2), e.g., automating word choices, human requirements remains crucial for customized or specialized tasks [38]. We define these task as *LLM-hard*, as a simple prompt cannot produce a satisfactory response. Developing reusable prompts for customized chatbots or GPTs [6, 77] is a common LLM-hard task. While a *prompt* is an input to LLMs, a *reusable prompt* is a set of instructions for recurring goals, essentially functioning as a *prompt program*. As more users create reusable prompts, the ability to write good prompt programs with clear requirements becomes increasingly important.

However, prompt creation can be complex and inaccessible to non-experts who lack technical knowledge of LLMs, highlighting the need for more end-user prompt engineering (EUPE) scaffolding and training [77]. Various strategies have been developed to support PE, such as prompt technique catalog [8], chat-based tools like GPT-builder,[2] and toolkit for orchestrating complex tasks [10, 29, 69]. Each tool has trade-offs, balancing flexibility, precision, and technicality. For example, chat-based tools may ask narrow clarifications without fully understanding the context, leaving users to self-refine requirements when LLM responses are open-ended [34, 51].

Existing PE training emphasizes the importance of explicitly stating requirements within prompts [16, 41, 57], yet there is still a lack of focused training on requirement generation skills for everyday users. Without training, non-experts often make mistakes such as missing requirements and writing conflicting requirements in prompts [15, 19, 44]. Even experts need many iterations to improve their requirements for complicated LLM tasks [59]. However, training novices on prompt engineering is difficult; for instance, novice programmers may not improve at prompting within a 75 minute study when provided with test cases and generated code as feedback [46]. We aim to address this gap by proposing a ROPE training for end-users to improve their requirement engineering skills in the context of prompt creation.

2.2 Instruction Following for Foundation Models

Optimizing LLM performance on customized tasks typically follows two approaches: improving the models directly, or refining prompts with models fixed. The former has enabled the versatility of models like GPT-4 [47], Gemini [14], and LLaMA 3 [3], which are fine-tuned through instruction tuning [48] and further aligned with human preferences using Reinforcement Learning from Human Feedback (RLHF) [13]. For the latter, the NLP community has been exploring LLMs' capabilities to follow requirements (or "constraints"). Various datasets have been proposed to assess whether LLMs can

---

[2]https://chatgpt.com/create

fulfill constraints, such as IFEval [80], INFOBench [53], and Follow-Bench [25]. These datasets focus on evaluating models' abilities to handle compositional constraints, and have demonstrated promising advancements.

Our work hypothesizes that requirements can be central to human-LLM interactions, as we have observed the LLM proficiency in following instructions. However, existing requirement datasets are often synthetic, containing prompts that are intentionally tricky or unrealistic, with requirements generated automatically via templates and limited to certain categories (e.g., sentence or paragraph length constraints, format constraints like output in JSON). While useful for identifying *LLM-hard* problems for requirement-focused training, these datasets do not fully capture how LLMs respond to real-world user prompts. Human prompts may have more diverse requirements across various tasks and less standardized language — a gap we seek to fulfill in this paper.

Building on this foundation of instruction following, *prompt optimizers* have been proposed to improve the performance of frozen LLMs on user-defined downstream tasks by automatically refining user-written prompts. Automation ranges from simple adjustments like adding role-playing or chain-of-thought prompting [66] (e.g., Prompt Maker in Figure 2) to more advanced methods like TextGrad [76], which searches for optimal wording, and DSPy [62], which extends the search space to few-shot prompts and various prompting techniques. While these sophisticated methods make more tailored changes, they are harder to apply and often require access to labeled datasets. Despite being in early stages, these optimizers aim to relieve humans from exhausting all possibilities in the natural language prompting space, as automated experiments with semantically-preserving edits can be more effective. We share their vision and foresee a future where optimizers play a crucial role in human-LLM interaction.

2.3 Requirements in Programming and Software Engineering

In software engineering and end-user software engineering, *requirements* describe what a human wants to achieve and how a program should behave in the world [31]. They encompass all the necessary conditions, constraints, and desired outcomes to ensure the output aligns with the human's needs and expectations. *Requirement engineering* stems as a field that focuses on generating and documenting requirements for software [64]. Good quality requirements need to be *accurate* and *complete*, without *commission* (inclusion of irrelevant or incorrect details) and *omission* (exclusion of necessary details) defects [4]. Previous studies have identified requirement engineering as a challenging skill to master. While training mechanisms have been developed to help students avoid commission and omission errors [45], a significant skill gap remains between graduates and professional engineers [18, 54].

Parallels can be drawn between prompt engineering (PE) and requirements engineering. For example, adapting LLMs to diverse scenarios demands well-specified requirements, and prompt authors often need to iterate on prompts that are too ambiguous for the LLM to interpret [50]. While PE training stresses the importance of clear requirements [16, 41, 57], this is often presented just as one of many prompting principles (e.g., one of 26 [8]), limiting its
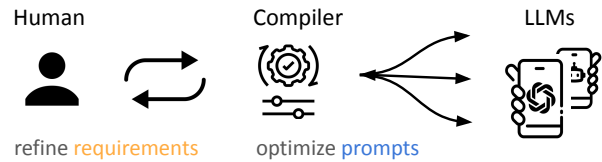

Figure 3: Our envisioned ROPE paradigm.

impact on prompt authors. In fact, little attention has been given to explicitly training end-users on requirement engineering in the context of prompt programs, including both natural-language-to-code and natural-language driven GPT applications. While several tools like EvalLM [30], SPADE [59], and EvalGen [60] have explored extracting user criteria to support prompt *evaluation*, the concept of requirements has not yet been fully emphasized in the context of prompt *construction*. We aim to address this gap by introducing the ROPE paradigm and developing a requirement-focused training mechanism to help end-users more effectively instruct LLMs to achieve their desired goals.

3 THE ROPE PARADIGM

We offer our definition of Requirement-Oriented Prompt Engineering (ROPE) and describe why, when, and who need it, and why we propose a training toward ROPE in this work. In short, ROPE represents a **paradigm shift in how we interact with LLMs, focusing on the importance of crafting accurate and complete requirements to achieve better results, especially for complex, customized tasks.**

The definition of requirements. We define a *requirement* as a skeleton instruction that communicates an essential condition or constraint on desired LLMs output (e.g., "Response is less than 100 words"). Requirements instruct LLMs to perform tasks that may deviate from their default behavior, guiding LLMs to align with user's goals.

In our work, we focus on evaluating requirements quality rather than quantity, and we operationalize requirement quality in our work (defined in Section 4.2) by adopting a taxonomy from the requirement engineering literature (Section 2.3). Ensuring requirements are accurate and complete is essential for effective LLM prompting, as poor-quality requirements can lead to harmful outcomes — e.g., missing requirement "anonymize the data" may make LLMs keep identifiable information in data, and vague requirement "delete harmful content" without a clear definition of "harmful" may cause LLMs to misinterpret sensitive topics like mental health or race as harmful.

Here we also focus on *natural language requirements* due to its universal understanding and tight connection to LLM prompting. While multi-modal requirements can be compiled into prompts for different LLMs, we leave this exploration for future work.

The relation between requirements and prompts. We view a prompt as a *super-set* of requirements. It contains not only users' customized requirements, but also other (orthogonal) factors like fluency and standard prompting tricks. Among these factors, requirements are more *user-centered* and *LLM-agnostic* — users' goals generally remain consistent across models (e.g., GPT-4, Gemini).

Fully articulating these requirements is essential, as omitted customized details can be difficult to recover automatically (explored further below). In contrast, other factors tend to be more *LLM-centered*: different models may prefer different writing styles, example types, or requirement order. While optimizing these factors can enhance LLM performance, they are better suited for automation rather than humans due to their (1) formulaic nature (e.g., wrapping prompts in a template consistently improves Llama-3-8b's performance on any task [2, 17]) and (2) idiosyncratic behavior (e.g., GPT-3 can be sensitive to semantically invariant edits that humans do not expect to make drastic differences [77]). This division is crucial given the rapid evolution of LLMs, as developing a mental model for a specific LLM can be a waste of effort in the long run.

With this distinction, we view ROPE as a task-delegation strategy. In crafting executable prompt programs, users should focus on *iterating and refining requirement-related components*, leaving other aspects automation. We find recent advancements in prompt optimizers (Section 2.2) particularly promising. With a consistent set of user requirements, we hope future optimizers could function like program synthesizers, searching through possible "implementations" — in terms of phrasing, examples, and structure — to generate prompts that best align with each LLM's preferences.

The applicability of ROPE: LLM-hard tasks and prompt programmers. In theory, any task can have countless requirements; for instance, implicit needs like "reply in the same language" are almost always assumed in LLM applications. However, not all requirements need to be explicitly stated. For standard tasks well-represented in LLM training data, e.g., "correct the grammar in this email", LLMs can meet expectations relying on their parametric knowledge without explicit clarification (e.g., on what grammatical rules exist).

We argue that explicitly articulating requirements becomes more crucial for what we refer to as **LLM-hard** tasks. These tasks require significant customization where users must intentionally *deviate LLMs from their default behaviors* [25, 53, 59], making autofulfillment impossible (e.g., how many rounds of questions should be asked in Figure 2). At the time of writing, tasks that involve multi-step processes, decision-making, or context knowledge are likely more *LLM-hard* and thus benefit more from explicit requirements. However, this is a dynamic concept that evolves as LLMs improve.

Nowadays, LLMs are widely used by *end user to create reusable prompt programs*, and these prompt programs are often LLM-hard, as users frequently seek customization with various requirements, anticipating diverse inputs and expected outputs. From content creators crafting prompts for creative graphics, educators tailoring LLMs as tutors, to office workers automating workflows with GPT, people across various fields are all building natural language prompt programs.[3] We argue that *all these potential prompt programmers would benefit from articulating better requirements.*[4]

The challenge of ROPE and the call for requirement-focused training. Ideally, we hope LLM users would naturally refine requirements in prompts based on unsatisfactory model outputs. In reality, this is challenging. Novices struggle to *understand what constitutes a requirement* in prompt engineering [77], while even experienced users struggle at *extracting and expressing requirements appropriately*. For example, users find it difficult to translate raw observations on model outputs ("I don't like how the chatbot didn't introduce itself") to clear requirements ("Introduce yourself at the start of the conversation, and state what you can help with") [36, 50]; Users also cannot decide on the right level of specificity and abstraction [37], e.g., use overly general keywords where more granular, domain-specific requirements are needed for LLM-hard tasks [44, 46].

We hypothesize that *users need explicit guidance to prioritize requirements during prompt creation.* Effective training should help users recognize the importance of clear, complete requirements and develop skills that connect abstract ideas with concrete model behaviors — goals we address in our training design.

4 TRAINING AND EVALUATION DESIGN FOR ROPE

We develop a **training and assessment suite** to help users improve their ability to write *accurate and complete* requirements for instructing LLMs. We adopt a backward design method [67], a well-established instructional design approach that starts by identifying the desired learning outcomes — in our case, writing effective requirements for LLMs — then works backward to develop assessments and training aligned with goals. Aligned assessments and training are critical to ensure that what is taught directly prepares participants for the skills they are expected to demonstrate.

Through multiple pilot studies with novices and experts ($n = 10$), we refine three key components of design: (1) realistic tasks that mirror real-world prompting challenges (Section 4.1); (2) assessments that connect requirement quality to LLM outcomes (Section 4.2); and (3) deliberate practices with feedback in a system (Section 4.3). Beyond training (Section 6), our ROPE assessment and materials also enable us to build more nuanced understandings on how requirements affect LLM outputs.

4.1 Task Design: LLM-Hard Prompt Programs for Replication

We aim for novices to eventually develop the ability to articulate requirements for their own prompt programs. However, to begin their training, we need to provide a set of concrete sample tasks for them to practice and for us to evaluate their progress. Below, we outline the six carefully designed tasks.

Task setup. To ensure the training and evaluation process is both realistic and representative, our primary objective is to have users *write natural language prompts to instruct LLMs in generating programs*. Instead of focusing on open-ended tasks — which are difficult to assess and provide consistent feedback on — we aim to *provide clear ground truths for evaluation and training*. To this end, users are asked to write prompts that instruct LLMs to **replicate a**

---

[3]While a prompt program can output code like traditional natural language programs [24, 55], it can also produce other outputs such as videos or GPTs applications. Note that when we refer to natural language program later in the paper, we are generally describing prompt programs.

[4]In contrast, more ad-hoc interactions with LLMs, such as casual conversations with the model, may require a different set of considerations, which we consider beyond the scope of this discussion.
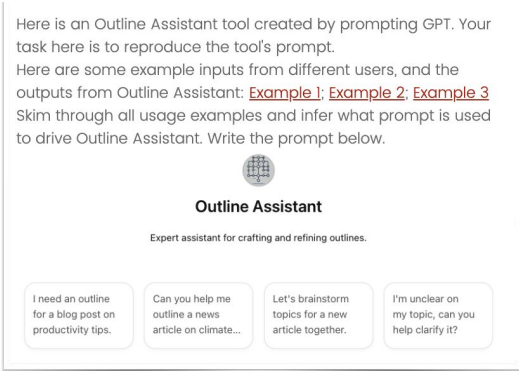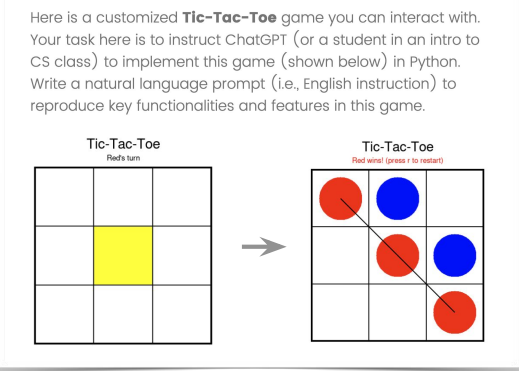
	GPTs Task		Game Task	
Task Type	Customized LLM powered directly by natural language		Natural Language to code (from CS assignment)	
Example & Requirements *(Rubrics)*	Here is an Outline Assistant tool created by prompting GPT. Your task here is to reproduce the tool's prompt. Here are some example inputs from different users, and the outputs from Outline Assistant: Example 1; Example 2; Example 3 Skim through all usage examples and infer what prompt is used to drive Outline Assistant. Write the prompt below. **Outline Assistant** Expert assistant for crafting and refining outlines. I need an outline for a blog post on productivity tips. / Can you help me outline a news article on climate... / Let's brainstorm topics for a new article together. / I'm unclear on my topic, can you help clarify it? 1. Use follow-up questions to let the users clarify and specify their needs, e.g., target audience, length, focus, tone, and style. 2. Offer advice for potential modification direction. (6 in total)		Here is a customized **Tic-Tac-Toe** game you can interact with. Your task here is to instruct ChatGPT (or a student in an intro to CS class) to implement this game (shown below) in Python. Write a natural language prompt (i.e., English instruction) to reproduce key functionalities and features in this game. Tic-Tac-Toe — Red's turn → Tic-Tac-Toe — Red wins! (press r to restart) 1. Render the title Tic-Tac-Toe on top of the board. 2. Display the corresponding message (e.g. Red's turn) 3. Key press 'r' to restart the game & reinitialize the board. (9 in total)	
Specificity	Higher-level requirements covering diverse category, but can have subjective interpretations		Lower level requirements akin to implementation specification but more verifiable	
Modality	Convey requirements through textual examples — users are given 2-3 chat histories as interaction examples. It is representative of *status-quo* prompt iteration.		Convey requirements visually — users are given an interactive game. The interaction reduces ambiguity that are common in example-based representations and requires less English reading skills.	

Table 1: An overview and comparison on the task types in our training.

given reference program. Each task consists of two key components: (1) A *gold reference program* with which users can interact with to deduce requirements; and (2) A set of *gold requirements* that precisely reflect the program's behavior. Ideally, an expert skilled in requirement-driven prompt engineering should be able to express all the gold requirements within their prompts, guiding the LLM to generate a program that mirrors the reference program's behavior exactly.

This setup requires users to derive requirements from provided examples. While different from generating requirements from scratch, it captures actual challenges of requirement articulation, especially for LLM-hard tasks in real-world (as described in Section 3): Users often have access to partial inputs or examples like a demo or a user story, often need to refine their requirements by analyzing available outputs, but do not always know how to connect model outputs with abstract requirements. We discuss potential future work of open-ended tasks in Section 7.1.

Task selection and customization. To cover broad types of realistic prompt programs users may write, we incorporate both GPTs applications directly powered by natural languages, as well as natural-language-to-code programs. We prioritize tasks most suitable for targeted and engaging training, through two dimensions: (1) we prioritize *generic tasks* over specialized ones (e.g., data science tasks), to ensure that the challenge lies in specifying requirements rather than applying domain-specific expertise, preventing construct irrelevance; and (2) we opt for *tasks with visual interfaces*, enabling users to interact with and visualize outputs in a manner that is accessible to novices.

With these criteria in mind, we select three GPT-driven tasks —Outline Assistant, Trip Advisor, and Email Proofreader —

along with three coding tasks presented as visual game interfaces: Connect4, Tic-Tac-Toe, and Tetris. All these tasks are inspired by real-world applications and have reference requirements, though we make necessary adaptations to ensure they are *LLM-hard* problems. The three game tasks are sourced from the Introduction to CS course at our institution, each with reference requirements provided by the instructor. They require minimal customization, as instructors have already incorporated LLM-hard requirements to prevent students from cheating using large language models. For the three GPTs tasks, we adapt real-world GPTs prompts,[5] and make them more LLM-hard by using frameworks like IFEval to introduce customized requirements such as format constraints [80]. We then derived the requirements by parsing the underlying prompts through expert annotation and pilot testing.[6]

While these tasks are aligned for the training and assessment design [67], they span a variety of application domains with unique requirements, preventing novices from relying on memorization and promoting meaningful learning. As shown in Table 1, the GPTs and Game tasks also offer complementary coverage on requirement specificity and modality.

Task validation. We distribute the six tasks in assessments and training session. We use Tetris and Email Proofreader in training, and the rest for pre- and post- assessments before and after training in a counterbalanced design. In pilot studies, we confirmed that (1) we could distinguish prompting novices and experts using the tasks, with the average novice performance being 28% and

[5]Writing Assistant prompt: link, Trip Advisor prompt: link, Email Proofreader prompt: link
[6]Detailed instruction to the task, as well as the requirement breakdown are in supplemental materials.

expert achieving full scores, (2) all tasks could be implemented by LLMs, with the average highest score being 91%, and (3) the assessment tasks were all comparably challenging for users.

4.2 Assessment Design: Requirement-Focused Intrinsic and Extrinsic Evaluation

To evaluate users' ability to instruct LLMs effectively, we assess both the quality of the user prompts (requirements within the prompt) and the prompt's impact on the quality of the LLM's output. Thus, for each user task completion, we calculate the *Overall Test Score* as is the average of the *Requirement Quality* and *LLM Output Quality* scores:

- *Requirement Quality Score*: This intrinsic metric assesses, "Does the user's prompt accurately and comprehensively cover all requirements?" It measures the completeness and correctness of the requirements by comparing those extracted from the user's prompt to expert-defined reference requirements for each task (described in 4.1). Drawing from the requirements defect taxonomy [4, 43], we track both *commission errors* (incorrect, inconsistent, or ambiguous requirements) and *omission errors* (missing ground-truth requirements). Given the inherent ambiguity in defining requirements and the varying levels of granularity (Section 3), we break the prompt into one-sentence clauses to extract individual requirements. The requirement quality is then operationalized as the percentage of requirement clauses that are free from commission or omission defects.
- *LLM Output Quality Score*: This extrinsic metric evaluates, "Can the user's prompt successfully guide the LLM to achieve the intended goals?" It measures the proportion of desired features implemented in the LLM-generated output that align with the reference (Section 4.1). We pass users' prompts to GPT-4o and compare the generated output to the reference program.[7] Since many features are difficult to evaluate automatically (e.g., GPTs behaviors cannot always be checked via rules), we rely on manual evaluation. For games, an expert interacts with the generated program to verify whether the requirement features are correctly implemented. For GPT interactions, an expert grades whether the GPTs' replies display expected behaviors (e.g., asking a follow-up question when the user's input is ambiguous). We grade two to three complete conversations per GPTs, with 5 turns of GPTs responses each.

Assessment validation. Three of the authors discuss to iterate the grading rubrics on the tasks during the pilot study. We confirm that expert can measure requirement quality using the percentage of correct requirement clauses in novice prompts, and that this correlates positively with the expert's judgement (Spearman's $\rho = 0.66$). For grading in the user study (Section 5), two authors (one of whom did not participate in the rubrics development during pilot) independently grade 10% of the randomly chosen pre-post test responses. We check the inter-rater reliability between the authors' scoring by calculating the Intraclass Correlation Coefficient (ICC) [33], finding a strong reliability ($ICC = 0.9$, 95% Confidence Interval $= [0.7, 0.98]$). Any discrepancies in scoring are resolved through discussion,

after which the authors individually grade half of the participants' responses. Detailed rubrics are provided in supplemental materials.

4.3 Interactive Training Mechanism: Dedicated Practice and Feedback on Requirement Defects

We design a training mechanism to support deliberate practice on requirement elicitation and refinement by *disentangling requirement articulation from peripheral tasks like persona crafting or paraphrasing in existing prompt engineering instructions.* We apply key learning principles like scaffolding and worked example [32], and we implement the training into an interactive interface as shown in Figure 4.

In this training, a user (a prompt novice) is asked to replicate one Game task (`Tetris`) and one GPTs task (`Email Proofreader`), *solely by describing the requirements of the given program as accurate and complete as possible.* For example, to reproduce the customized `Tetris` game in Figure 4, the user will first start by outlining the main milestones (e.g., creating the game board, handling piece placement, etc.), and then provide more detailed specification per step (e.g., define the size of the game board). Novices are not expected to achieve full success on their first try; we use three types of **requirement-focused feedback** to guide users in continuously refining their requirements:

- *Textual hint and clarification*, via chatbot (Figure 4A): We create a tutor chatbot to offer users a natural, conversational experience to discuss and reflect on their requirements. For instance, the chatbot may ask "What's on top of the board?", when a novice misses the requirement for "Tetris title rendering". The textual feedback encourages critical thinking on *missing* or *incorrect* requirements.
- *Reference requirement example*, via requirement working document (Figure 4B): We progressively reveal reference requirements when users mention them during interaction with the chatbot. The expert-written reference examples provide *reinforcement on correct requirements*, helping users understand how to formalize and organize requirements.
- *LLM output counterfactual*, via generated visualization (Figure 4C): For incorrect requirements that can be visually demonstrated, we generate flawed programs by implementing *incorrect* requirements (e.g., wrong board dimensions), or maliciously misinterpreting *ambiguous* requirements. For example, a vague requirement like "Use keys to move pieces" might result in a flawed Tetris game where pieces can move upward, exposing the unspecified allowed movement in requirement. The visual counterfactual is currently limited to the controllable game code generation; as GPTs outputs tend to be less predictable, we display static chat histories as illustrative examples for GPTs.

Note that our assessment tasks require users to write complete prompts similar to those in Figure 2, we deliberately avoided having users write prompts during training. Instead, users engage in conversational interactions designed to continuously encourage them to think about requirements. Additionally, users' written responses do not directly trigger LLM output generations. We use reference requirements to guide the LLM generation, ensuring that all feedback
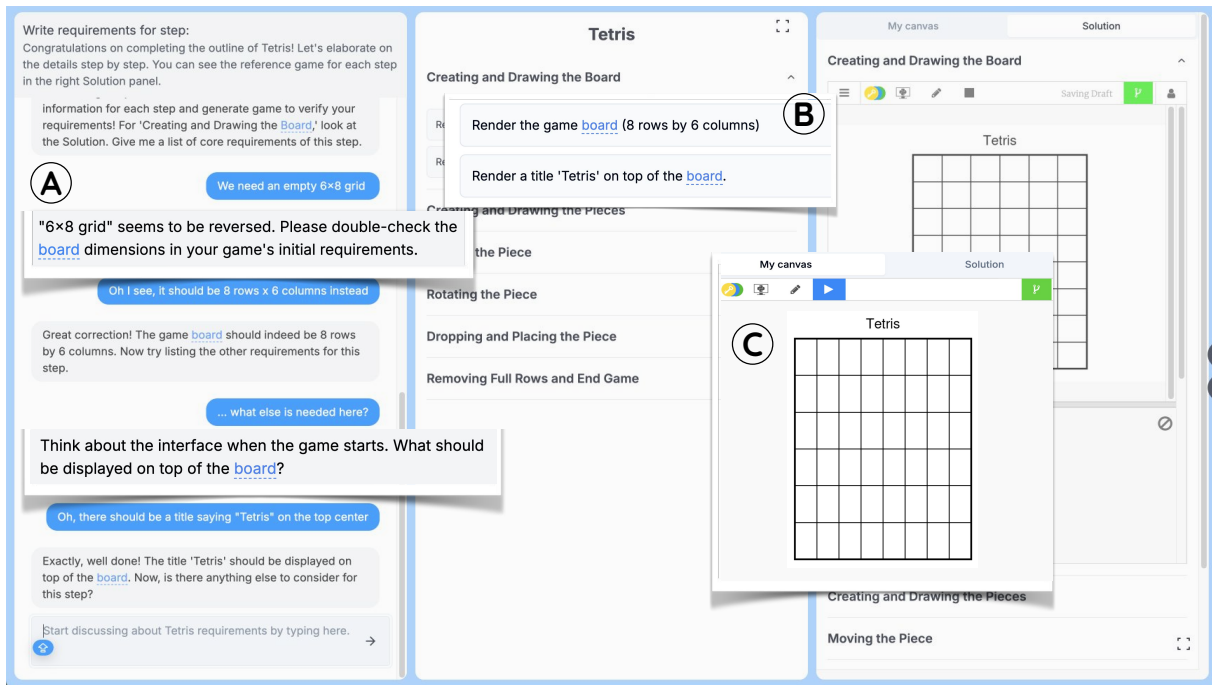
---

[7]The output is generated using a prompt similar to "Follow this prompt: {user_prompt}". Further details are in the supplemental materials.

Figure 4: Our ROPE training interface, with three types of feedback on requirement defects: (A) *Conversation-based hints* **on incomplete or inaccurate requirements, (B)** *Reference requirement examples* **to reinforce appropriately identified and expressed requirements, and (C)** *LLM output counterfactual* **to support user's reflection on incorrect or ambiguous requirements.**

reflects the quality of the requirements alone. This approach ensures that novices remain focused on improving their requirement articulation skills, without distractions from other factors.

Interaction and feedback implementation. We use OpenAI GPT-4o to power the interface,[8] leveraging its interactive nature to adaptively provide feedback on the diverse set of possible user inputs. However, feedback generation does not rely on the model's inherent reasoning capabilities but is anchored in our predefined reference requirements. Specifically, we always have GPT-4o to compare the current user requirements to the reference, and select the most critical defect to provide feedback on (roughly, mismatched requirements are considered most critical, then missing requirements, then ambiguous ones). Then, we have the LLM respond targetedly to the defect. For example, visual counterfactual examples on game tasks are generated by having the LLM minimally edit reference Python code based on the identified incorrect requirement. To enhance interaction, we also implement features like highlighting special variables (e.g., board, keystrokes) in conversations and as hyperlinks for easy cross-referencing across the interface.

Validation on feedback generation. We analyze the interface log data to evaluate the feedback quality In our user study (Section 5), we collected a total of 635 interaction turns from `Tetris` (ranging from 11 to 73 turns per user) and 180 turns from `Email Proofreader` (ranging from 4 to 20 turns per user). From this,

we randomly selected 10 different users and annotated conversations for 5 users per task, resulting in 151 `Tetris` turns, 66 `Email Proofreader` turns, and a total of 113 annotations per feedback type. For each LLM turn, we assess: 1) Is the feedback needed? 2) Is the feedback provided? 3) Is the feedback right? Each feedback was categorized as correct, incorrect, irrelevant, missing, or not provided. Note that the correct feedback includes both feedback that was correctly provided when needed, and feedback that was correctly *not* provided when unnecessary.

Two authors independently annotated one participant's chat log for each task, and we measured Inter-Rater Reliability (IRR) by calculating Krippendorff's α [9], achieving a high agreement rate ($\alpha = 0.87$) across all feedback types (see Table 2 for the detailed breakdown). Any discrepancies were discussed and resolved, after which one author completed the remaining annotations. The final analysis revealed a correct feedback rate of 88.6%.

5 USER STUDY DESIGN

We conducted a user study to understand whether our requirement-focused training is effective in improving novices on writing requirements in prompts, and whether requirement-focused training is *more* effective than *status-quo* prompt engineering training. We also examined how automatic prompt optimization affects performance.

Study procedure. We employed both within-subject and between-subject designs in our study. To capture the requirement-focused training gains, we utilized a **pre-test and post-test approach**, a

---

[8]The state-of-the-art LLM at the time of writing: https://openai.com/index/hello-gpt-4o/. We used a temperature of 0.3 for code generation and 0.7 for other tasks. Refer to supplemental materials for our prompts.

Table 2: Proportions of feedback types across different feedback sources and IRR (Krippendorff's α) of human annotations

Feedback Source	Right Feedback			Wrong Feedback				IRR (α)
	Correct	Not Provided	Total	Incorrect	Irrelevant	Missing	Total	
Text	69.03%	20.35%	89.38%	10.62%	0.00%	0.00%	10.62%	0.86
Requirement Doc	48.67%	38.05%	86.73%	1.77%	5.31%	6.19%	13.27%	0.88
LLM Output	3.75%	86.25%	90.00%	0.00%	5.00%	5.00%	10.00%	0.68
Total Feedback	44.44%	44.12%	88.56%	4.58%	3.27%	3.59%	11.44%	0.87

standard instruction evaluation method [56], comparing participants' performance on isomorphic tasks before and after training. To compare requirement-focused training with standard training, we utilized a **randomized-control design**. We randomly assigned participants to either the experimental condition (ROPE group), where they receive requirement-focused training in our interface, or the *control* condition (PE group), where they receive a standard prompt engineering tutorial and then self-practice using ChatGPT.

Our user study — for either the ROPE or the PE group — is 1.5 hours long. In the study, participants first completed a two-minute *pre-survey*, providing demographic information and rating their prior prompting experiences with 7-level Likert Scale questions. Then, they started a *pre-test* (20 minutes) to write prompts to replicate key features on existing programs (Table 1) — one Game (Tic-Tac-Toe or Connect4) and one GPTs (Outline Assistant or Trip Advisor). They proceeded to the 40-minute *training session* (different interfaces depending on experiment group) on two tasks (Tetris and Email Proofreader), followed by a *post-test* that contained different Game and GPTs tasks (20 minutes). To further capture participants' iterative prompting behaviors, for the post-test GPTs task, participants were also asked to feed their prompt into ChatGPT and iterate their original prompt by observing the ChatGPT outputs. They completed the study with another two-minute *post-survey*, providing feedback with open-ended or Likert Scale questions on their experience and perceptions during the training. Participants were compensated with a $20 Gift Card.

Control group (PE) design. We designed the PE group to replicate the "business-as-usual" scenario, where novices learn prompt engineering through publicly available resources and self-practice. Specifically, PE group participants first watched a 20-minute YouTube tutorial,[9] covering best prompting practices including clear instructions, adopting personas, specifying format, limiting the scope, and few-shot prompting. The tutorial included various prompting examples such as poem writing, code-based data processing, essay summarizing, etc. Afterward, participants self-practiced writing prompts for the two training tasks, iterating with ChatGPT directly. For the game task, we offered participants an online code compiler so participants could see games rendered in real-time, similar to the visual feedback in our interface (Figure 4C). Similar to the ROPE group setup, our PE group offers practices on coding and GPTs tasks to establish task familiarity. However, unlike our requirement-focused training where users practice writing only requirements without actually writing prompts, the conventional practice for PE group directly train participants on writing prompts, which is closer to what users actually write in pre-post tests, and the direct

interaction with ChatGPT might help users build a better mental model on LLM behaviors.

Data collection. We collected participants' responses (prompts), time on task, as well as interaction log data during the training phase. Afterward, we collected the LLM outputs by feeding participant prompts into OpenAI GPT-4o.[10] Using the evaluation method and rubrics we developed for each of the assessment tasks Section 4.2, we graded the pre- and post- tests using users' *original prompts* for all tasks from the test, on a scale of 0% to 100%. We calculated the *Overall Test Scores* as the average of the *Requirement Quality* and *LLM Output Quality* scores. The pre- and post-assessments each contained one Game task and one GPTs task (Section 4.1), counterbalanced to mitigate problem sequencing bias or different task difficulties. We found that the two versions of the tests have comparable difficulties from pre-test results (an average of 22.9% and 18.3% for each version).

To understand the effect of an optimizer on prompt (as discussed in Section 2.2), we further use the Prompt Maker,[11] a popular optimizer with 50k usages in GPT Store, to refine participants' *original prompts* into *optimized prompts*. This optimizer is easily applicable to any kinds of prompts (Figure 2 shows an example of its output), though it is a rather preliminary version, as it rephrases user prompts in a rule-based manner without criticizing the LLM outputs of the original prompt. We evaluated the optimized prompts in the same way as the human prompts (Section 4.2), including generating and grading corresponding LLM outputs.

Participants. We recruited users with limited or no NLP background from different institutions in the United States. We conducted studies with 32 participants randomly assigned to the ROPE or PE group, and we removed two participants who disengaged during the study from the analysis. In total, we have 30 participants (S1-30, $n = 15$ for each condition) — 19 female, 10 male, 1 agender, and 18 non-native English speakers, with an average age of 26.

6 USER STUDY RESULTS

6.1 Learning Gain: Requirement-focused Training Helps Students Instruct LLMs

We start by answering our central research questions: *Can we help end-users better instruct LLMs through requirement-focused training?* We quantitatively measured learning gains by capturing participant performances from the pre-test to post-test within each

---

[9]Relevant sections from Prompt Engineering Tutorial – Master ChatGPT and LLM Responses

[10]We used default settings such as a temperature of 1, please refer to our supplemental materials for more details.
[11]https://chatgpt.com/g/g-hhh4w3eov-prompt-maker: from a simple prompt to an optimized prompt.

experiment group, and evaluated the effectiveness of the two training approaches by comparing the ROPE and PE group. We further unpacked the quantitative results through analyses of participants' Likert Scale ratings, as well as the transcribed comments during their training session and in post surveys.

Requirement-focused training is effective. Using a two-tailed paired t-test,[12] we found that test scores for participants in the ROPE group significantly improved by 19.8% from pre- to post-test ($p < 0.001$, from 20.0% ± 14.6% to 39.8% ± 18.8%), a two-fold increase. This demonstrated that *requirement-focused training enabled participants to instruct LLMs more effectively,* and we achieved our desired goal: *to concentrate users' attention on only requirement iterations during the training, but make sure users can still apply their learning in overall prompt writing.* Analyzing requirement defects in ROPE participants' prompts, we observed a noticeable decrease in omission errors (from 5.6 to 3.2 per participant), but a slight increase in commission errors (from 0.5 to 0.7 per participant). This indicated that participants *became more aware of requirements, but need additional training for requirements clarity and accuracy.*

ROPE participants' self-reflection highlighted that they understand the value of *emphasizing and iterating* on requirements, which may contribute to their improvement. Six ROPE participants explicitly noted the connection between requirements and prompts in their post-survey. For example, S30 described requirements as a way "to be more clear and not confuse the system by giving too much unnecessary information", and S8 noted that a requirement-focused approach helped them *organize their thoughts*: "I learned to organize my requirements logically so we can easily revise and improve them". S10 neatly described their shift towards requirement-focused prompting strategy: "sometimes when I write prompts, [I find] steps are hard to be clearly divided, or I didn't consider to divide them that detailed. However, it's important to do so, as it appears to give LLM more direct and clear instruction. When the steps are divided, it's easier to see the missing details in my original prompts too".

Requirement-focused training is more effective than standard prompt engineering practice. From the post-test prompt data, we observed that requirement-focused training and standard practice had distinct effects on how participants wrote prompts. ROPE group spent more time (19 minutes vs. 14 minutes) and wrote longer prompts (796 vs. 458 characters) than the PE group. In 100 randomly selected pairwise comparisons, ROPE participants produced more structured prompts 87% of the time, consistent with their self reflections reported above.

These prompt also revealed learning gain differences. A two-sample t-test showed that the ROPE achieved significantly higher learning gains compared to the PE group ($p < 0.001$). In fact, while ROPE significantly improved (19.8% as mentioned above), we did not observe much gain in PE group's pre-to-post score (1.2%). The contrast suggested that *novices indeed could not acquire requirement articulation skills through standard prompting training alone.*

Participant feedback supported our hypothesis that *requirement articulation skills do not naturally emerge.* While we hoped interacting with LLMs would help participants build mental models of LLM behaviors, PE participants found the LLM response too unpredictable for effective self-practice. Similar to prior work reporting the challenges of prompting nondeterministic LLM [46, 77], many got frustrated during their unfruitful self-practice sessions: "I was confident with my communication ability at first but later during the tasks I was frustrated with my communication skill with AI" (S11). The unpredictability distracted participants from applying the best practices taught in the tutorial, as S19 noted: "I lost some of the concepts at the end that looked very minor in determining the effectiveness of the prompt".

Requirement-focused training encouraged more targeted and iterative prompting. The aforementioned pre- and post-tests analyses revealed participants' learning gains in *writing initial prompts for natural language programs*. Beyond initial prompts, we analyzed participants' prompt iteration behaviors on the GPTs tasks at the end of post-tests. We observed that *requirement-focused training led users' to make more requirement-related changes during iterations.*

In the PE group, most participants either chose not to make any iterations (5 out of 15) or only made superficial edits (e.g., altering wording or grammar, 8 out of 15), with only 2 adjusting prompt requirements. In contrast, after ROPE training, participants *were more likely to engage in substantive requirement engineering*, updating or adding specific requirements instead of making random edits typical of end-user prompt engineering. Eight of 15 ROPE participants made meaningful requirement adjustments (5 were fully successful, 2 made partial progress, and 1 introduced an incorrect requirement). Participants were also aware of their requirement-centric iterations: "[My strategies were to] break down my requirement into several key points, use examples, iterate, self-check if more details are needed, or if more steps should be elaborated, ..., see the test result using examples" (S10). Overall, ROPE training effectively equipped users with a more structured and systematic approach to prompt generation.

Key to success: deterministic feedback as a jump start. Looking at both learning gains and behavioral changes, we speculate that the success of requirement-focused training stemmed from our use of *requirement-focused feedback.*

Requirement articulation was challenging for all participants, as shown by the low pre-test scores (20% for the ROPE and 23% for the PE). This poor starting point likely hindered the PE's ability to receive useful feedback during self-practice, as their requirements were too weak for the LLM to generate "good enough" model output with clear indicators on what to improve. Consequently, PE participants were trapped and frustrated by LLM unpredictability — S6 noted that "ChatGPT is a very malleable tool and can change responses pretty drastically depending on the prompt", and S7 commented that "if ChatGPT doesn't get it, I might not be patient".

In contrast, our feedback loop for ROPE (Section 4.3) provided tightly controlled *feedback* and *program output* based on the requirements users specified, regardless of grammatical issues or natural language variances. S8 highlighted that: "It's easy to understand (very logical) and can see how the changes and revises

---

[12]We did a linear mixed effects regression to account for the correlation across participants and task (both as random effects) of the repeated measures for the pre- and post-test times and for the requirement and LLM output scores. We found the same pattern of significant results as revealed by the t-tests.

influences the system immediately". This deterministic feedback reinforced ROPE participants' believes on the importance of clear requirements, leading to them writing better initial prompts with more requirements in the post-test. We hypothesize that these better initial prompts might also become *more suitable for further iteration*. Although still imperfect, ROPE participants' prompts might allow the LLM to generate partially correct outputs that contain enough signal towards future improvements.

We suspect that effective prompt engineering training does require deterministic scaffolding. Novices may need reinforcement outside of LLM feedback until they reach a level where interpreting LLM responses becomes valuable.

6.2 In-depth Analysis: The Validity of ROPE Paradigm

Going beyond learning gains, we dive deeper into the connections between requirement in prompts and LLM output quality, as well as the role of optimizers.

Requirement quality vs. LLM output quality: promising correlation with nuances. To examine whether more correct and complete requirements led to better LLM outputs, we calculated the correlation between the two components of our *Overall Test Scores*: *Requirement Quality Score* and *LLM Output Quality Score*. We found a strong positive correlation, with a Spearman's correlation coefficient of $\rho = 0.72$.

However, task-specific analysis revealed nuances. As shown in Figure 5, while all other tasks achieved $\rho \geq 0.88$, Connect4 did not show any correlation. Besides one ROPE participant who scored 80% on requirement quality and 70% on LLM output in their post-test Connect4, all other participants' LLM output scores saturated around 40% even with high requirement quality $\geq 80\%$.

Upon reviewing user prompts and LLM outputs, we suspected that Connect4 had fewer natural-language-to-code mappings in GPT-4o's training data.[13] Consequently, the model encountered difficulties implementing uncommon requirements in code for Connect4, such as "Cross out the winning cells", which only succeeded in 1 out of 16 prompts. Moreover, as shown in Figure 5, Connect4 prompts formatted as code implementation plans produced better results than casual narrative prompts, even with the same number of requirements. After we reformatted all game prompts into similar formats with the optimizer, the correlation between requirement and LLM output for Connect4 increased from $\rho = 0.02$ to $\rho = 0.3$, partially supporting our hypothesis. In contrast, Tic-Tac-Toe did not exhibit such sensitivity, likely due to a broader variety of rules in LLM's training data. GPTs tasks did not have this problem since both the prompt and implementation are in natural language.

We further analyzed how different types of requirement defects impact LLM output quality. We found that the numbers of omission errors (incomplete requirements) had a stronger negative impact on *LLM Output Quality Score* ($\rho = -0.4$) compared to commission errors (inaccurate requirements) ($\rho = 0$). This suggests that while LLMs can correct inaccuracies in requirements, they struggle to

fill in missing information — further highlighting *the importance of training humans to express all their necessary requirements*.

Our findings complement existing existing instruction following research [25, 80], which tend rely on synthetic prompts with perfectly written requirements of a fixed set of common constraint types (Section 2.2). Our results highlight the need for further investigation into how imperfect user requirements — both in content and format — interact with inherent LLM biases.

Optimizer improves prompts but introduces biases too. Finally, we evaluated how the optimizer affected prompt quality, experimenting on a single optimizer to explore feasibility, without generalizing to all optimizers (Section 7.1).

We first noticed that the Prompt Maker optimizer effectively enhanced user prompts by adding elements such as role-playing and chain-of-thought, making prompts more structured and fluent, as shown in Figure 2. Besides formatting, the optimizer also made direct modifications to requirements; on average, it added 0.5 missing requirements and corrected 0.3 inaccurate requirements, reducing omission errors from 4.4 to 3.9 and commission errors from 0.6 to 0.3 per participant across all prompts. These changes significantly improved overall test scores, as shown by a paired t-test for all users' prompts before and after optimization ($p < 0.001$).

It seemed like there exist some requirements that were *LLM-hard* but not *optimizer-hard*. For example, GPT-4o would not implement "Display messages for a tie game" unless explicitly prompted, but the optimizer would automatically add this requirement to user prompts. However, highly customized requirements, such as "Cross out the winning cells", were never automatically added (0 out of 51) without explicit user input.

Importantly, *the optimizer could not close the performance gap between the* ROPE *and* PE *group*. We computed the gains of PE participants after using the optimizer (8.8% = optimized prompts' post-test scores − original prompts' pre-test scores). Using a two-sample t-test, the ROPE's 19.8% pre- to post-test gains remained significantly higher ($p = 0.01$) than PE group's optimized gains (8.8%). This supports the finding that missing, highly customized requirements are unlikely to be added automatically.

While the optimizer generally had a positive impact, we also observed instances where *it misinterpreted user inputs or added irrelevant details*. For instance, for four out of five participants who requested an alphabetic list (A, B, C), the optimizer changed it to a numbered list. In some cases, these misinterpretations drastically reduced LLM performance. For example, two participants used the word "interactive" in their prompt, leading the optimizer to incorrectly add the role of "intensive interactive web application developer" and request the use of "Flask and HTML" in the optimized prompts, leading to poorly implemented code since it is much harder to generate a functional web app.

These issues highlight *gaps in user-expressed requirements and LLM-received requirements*, suggesting that future training should address the risks of misinterpretation by either the LLM or the optimizer (Section 7.2). On a more positive note, such misinterpretations could provide useful feedback in the future, alerting users to potential misalignment between their requirements and the LLM's understanding.

---

[13]Confirming these hypotheses would need more systematic probing and quantification [52], which we defer to future work.
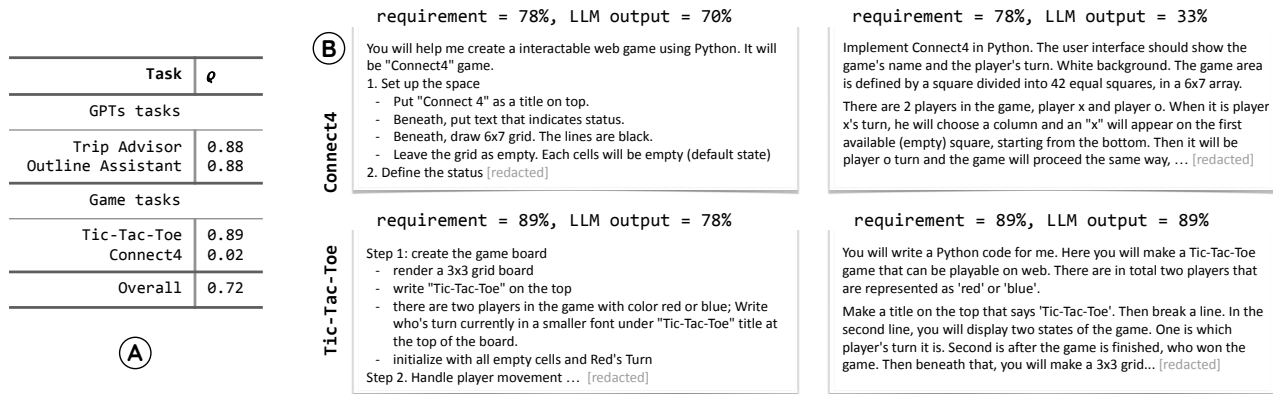
Figure 5: (A) Spearman correlation (ρ) between LLM Output and Requirement Score by Tasks. (B) A comparison between the prompt format sensitivity of `Connect4` and `Tic-Tac-Toe`. We suspect the low correlation `Connect4` on requirement vs. LLM output quality is due to the lack of natural-language-to-code mapping in LLM training on this task, making it more sensitive to prompt formats.

7 DISCUSSION

In this work, we introduce Requirement-Oriented Prompt Engineering (ROPE), a human-AI collaborative prompting paradigm where humans focus on effective requirements specification. Our evaluation shows that requirement-focused training significantly enhances novices' ability to extract and articulate requirements in prompts, leading to more goal-aligned LLM outputs; we also notice mismatch between humans' under-specified requirements and LLMs' misinterpretations is a key communication barrier (Section 6.2). While our work made an important step toward ROPE, several limitations remain, and there are many interesting questions to explore. Here, we reflect on possible immediate extensions on requirement-focused training, as well as the longer-term evolution of the ROPE paradigm.

7.1 Limitations and Next Steps on Requirement-Focused Training

Our training program significantly improved participants' ability to extract and articulate requirements, though the skill remains challenging. Post-training assessments showed an average score of 40%, with the top performer increasing from 24% to 63%. To further improve the effectiveness of requirement-focued training, we propose several next steps.

One easy extension is to make the training session longer. We observed that the 40-minute training time was difficult for non-native English speakers, and participants had limited opportunities for iteration due to time constraints. Future work should explore longer training sessions and adaptations for users with varying language proficiencies.

In addition to extending length, broadening the scope of the training will help improve the generalizabilty. Our study showed that the skills gained transferred to new GPTs and game development tasks. However, an open question remains: how well do these skills apply to other tasks like data analysis or creative content generation? Fortunately, our ROPE interface is adaptable to different tasks, as training can be generated based on provided reference requirements and LLM outputs. We plan to open-source ROPE and encourage its use for customized tasks across domains.

Besides length, our training can also be broadened for better generalzability. Our study demonstrated that the requirement specification skills users acquired through training generalized to new GPTs and Game development tasks. However, an open question remains: how broadly do these requirement skills transferred to other natural language programming tasks (e.g., data analysis or creative content generation)? Fortunately, our ROPE interface is adaptable to different tasks, as the training and feedback loop can be easily generated based on provided reference requirements and LLM outputs. We plan to open-source ROPE and encourage its use for customized tasks across domains. Moreover, beyond human-AI interactions, ROPE skills — such as requirement extraction, iterative refinement — can be broadly applicable to fields like software engineering and design. Future research should explore further transfer tasks in assessments, and also how ROPE skills may enhance communication, problem decomposition, and computational thinking across various fields.

Another dimension of generalizability is towards users' different prompting behaviors. Our study focused on reverse engineering fixed requirements, which allowed for controlled feedback but did not fully capture real-world prompting scenarios: users often engage in one-off interactions (e.g., direct question-answering) or iterative and open-ended prompting (e.g., chatting with LLMs on a rough idea), etc. [30, 59]. Future studies should investigate more self-directed tasks, compare different types of prompting, and tailor training accordingly. While providing feedback without clear requirements may be challenging, pre-generated materials on common novice approaches could offer useful scaffolding.

One primary bottleneck for broadening the scope is our lack of automatic assessment methods for the LLM-hard tasks. Our time-consuming manual grading effort limited our ability to conduct more experiments by e.g., grading multiple LLM outputs per user prompt for stability, experimenting with multiple optimizers, etc. Future work should explore automatic assessments and their trade-offs. For example, LLM-as-a judge paired with few-shot prompting

might provide reasonable estimations [79], and might be suitable for tasks that contain substaintial numbers of requirements where wrong gradings on a single requirement dot not drastically affect the assessment results.

7.2 Reflection on the ROPE Paradigm

Along with prior work on requirement-oriented evaluation [30], we believe that ROPE moves us towards a future where **requirements serve as the central interface between AI and humans.** Here, we discuss key skills humans and LLMs need to develop to prepare for a ROPE future.

What should humans be equipped with for ROPE? end-to-end prompting skills. To ensure success of ROPE, multiple skills need to be cultivated among humans. Our training successfully helped users develop the ability to *extract requirements from given examples and express them completely and correctly*, but our study also revealed more opportunities in requirement iteration and testing skills of an end-to-end prompting procedure. Specifically, users need to identify examples to compare LLM outputs and adjust the requirements accordingly.

For example, users should *create diverse and representative examples to identify and test requirements*, particularly for ambiguous or open-ended tasks involving edge cases or complex interactions. In our study, many users, especially those unfamiliar with games like `Connect4`, missed the testing scenarios like a tie game, which led them to omit the requirement "Display a "Tie Game!" message if no player wins" in their prompt. This is consistent to findings in debugging training [39] and other end-user prompt engineering work [77], as novices often create too few test examples. Future studies should support users to generate more testing examples.

Additionally, users should *iterate requirement granularity based on task "LLM-hardness."* This involves observing LLM failures and refining requirements, essentially developing a theory of mind for LLMs [65]. Some of our participants was already thinking about "what is obvious enough for ChatGPT to already understand/perform well on" (S28) and adjust their prompt accordingly (although not necessarily correctly). In our training, we introduced the concept of requirement specificity in `Tetris`, where users generate requirements in two levels of granularity (i.e., main steps and details, Section 4.3). However, some tasks required even greater specificity, such as generating non-standard `Connect4` gameplay elements "Cross out winning cells". This ability to adjust specificity in requirements can be influenced by domain expertise, for example, expert developers can iterate prompts with more accurate details to successfully generate code while students often struggle [44]. Future studies should offer adaptive support for adjusting requirement granularity based on task complexity and user expertise.

What should optimizers and LLMs be capable of for ROPE? support complementary task delegation. For a successful ROPE future, optimizers and LLMs must develop complementary capabilities to support human skills.

First, as humans improve on articulating good requirements, optimizers should *test different "implementations" of requirements*. This includes varied wordings, alternative expressions (e.g., zero-shot descriptions or few-shot examples), and the order, hierarchy,

or strictness of requirement compositions. Optimizers should also dynamically group the same requirements presented in multiple formats to reveal which (combinations of) requirements are missing or ambiguous.

Second, optimizers can also *assist test data synthesis*. More advanced optimizers need to iterate prompts on validation datasets and automatic objective functions, similar to machine learning. Insights from software engineering test automation [28] or NLP approaches to extract or synthesize test cases from natural language requirements could be helpful [22, 70, 71, 74].

Finally, LLMs or optimizers should *explicitly communicate their interpretations and reveal biases in their implementation*. As we observed in our study, optimizers can misinterpret requirements or add unnecessary details (e.g., "interactive game" can lead to unwanted features like Flask integration). LLMs should explicitly communicate their understandings of user intent, such as translating ambiguous natural language requirements into more formal specifications [37, 73], asking clarifying questions [11, 34, 51], or extracting and refining generated criteria [30, 75]. This is not a perfect solution, as models' translations may still reflect their biases and models may miss to ask about *LLM-hard* requirements. Nonetheless, to correctly implement requirements in LLM outputs when what is *LLM-hard* keep evolving, humans need to learn when and how to specify requirements with more granularity, and models need to keep communicating what requirements might be *LLM-hard* or *LLM-conflict*.

8 CONCLUSION

In this work, we advocate for focusing prompt engineering effort on human-centered tasks, ensuring users include all necessary requirements in their prompt to achieve goals. We introduce the Requirement-Oriented Prompt Engineering (ROPE) paradigm, and design training materials where prompting novices practice requirement articulation on complex prompting tasks. We also develop aligned assessment metrics capturing both intrinsic quality of user prompts in terms of requirement quality, as well as the extrinsic quality of prompt effectiveness in achieving intended outcomes. By providing targeted feedback on requirements, our ROPE system helps users produce higher-quality requirements and prompts more effectively than traditional prompt engineering training. As we look to the future, it is clear that the demand for LLM-based applications will only grow. As LLMs become more integrated into complex task-solving for more users, the ability to clearly articulate requirements will be key to effectively guide LLMs. We believe that users should be equipped with the foundational skills to prepare for the ROPE future, and the right training will empower users to harness the full potential of LLMs.

REFERENCES

[1] [n. d.]. Introducing the GPT Store. https://openai.com/index/introducing-the-gpt-store/. Accessed: 2024-9-2.

[2] [n. d.]. Llama 3. https://llama.meta.com/docs/model-cards-and-prompt-formats/meta-llama-3/. Accessed: 2024-9-11.

[3] Meta AI. 2023. LLaMA 3: Achieving More with Less Data. (2023).

[4] Amira A Alshazly, Ahmed M Elfatatry, and Mohamed S Abougabal. 2014. Detecting defects in software requirements specification. *Alex. Eng. J.* 53, 3 (Sept. 2014), 513–527. https://doi.org/10.1016/j.aej.2014.06.001

[5] Hannah McLean Babe, Sydney Nguyen, Yangtian Zi, Arjun Guha, Molly Q Feldman, and Carolyn Jane Anderson. 2023. StudentEval: a benchmark of student-written prompts for large language models of code. *arXiv preprint arXiv:2306.04556* (2023).

[6] Stephen H Bach, Victor Sanh, Zheng-Xin Yong, Albert Webson, Colin Raffel, Nihal V Nayak, Abheesht Sharma, Taewoon Kim, M Saiful Bari, Thibault Fevry, Zaid Alyafeai, Manan Dey, Andrea Santilli, Zhiqing Sun, Srulik Ben-David, Canwen Xu, Gunjan Chhablani, Han Wang, Jason Alan Fries, Maged S Al-shaibani, Shanya Sharma, Urmish Thakker, Khalid Almubarak, Xiangru Tang, Dragomir Radev, Mike Tian-Jian Jiang, and Alexander M Rush. 2022. PromptSource: An integrated development environment and repository for natural language prompts. *arXiv [cs.LG]* (Feb. 2022). arXiv:2202.01279 [cs.LG] http://arxiv.org/abs/2202.01279

[7] Tom B Brown. 2020. Language models are few-shot learners. *arXiv preprint arXiv:2005.14165* (2020).

[8] Sondos Mahmoud Bsharat, Aidar Myrzakhan, and Zhiqiang Shen. 2023. Principled instructions are all you need for questioning llama-1/2, gpt-3.5/4. *arXiv preprint arXiv:2312.16171* (2023).

[9] Santiago Castro. 2017. Fast Krippendorff: Fast computation of Krippendorff's alpha agreement measure. https://github.com/pln-fing-udelar/fast-krippendorff.

[10] Harrison Chase. 2022. *LangChain*. https://github.com/langchain-ai/langchain

[11] John Chen, Xi Lu, Michael Rejtig, David Du, Ruth Bagley, Michael S Horn, and Uri J Wilensky. 2024. Learning Agent-based Modeling with LLM Companions: Experiences of Novices and Experts Using ChatGPT & NetLogo Chat. *arXiv [cs.HC]* (Jan. 2024). arXiv:2401.17163 [cs.HC] http://arxiv.org/abs/2401.17163

[12] Bhavya Chopra, Ananya Singha, Anna Fariha, Sumit Gulwani, Chris Parnin, Ashish Tiwari, and Austin Z Henley. 2023. Conversational Challenges in AI-Powered Data Science: Obstacles, Needs, and Design Opportunities. *arXiv [cs.HC]* (Oct. 2023). arXiv:2310.16164 [cs.HC] http://arxiv.org/abs/2310.16164

[13] Paul F Christiano, Jan Leike, Tom Brown, Miljan Martic, Shane Legg, and Dario Amodei. 2017. Deep reinforcement learning from human preferences. In *Advances in neural information processing systems*. 4299–4307.

[14] Google DeepMind. 2023. Gemini 1 Technical Report. (2023).

[15] Paul Denny, Juho Leinonen, James Prather, Andrew Luxton-Reilly, Thezyrie Amarouche, Brett A Becker, and Brent N Reeves. 2024. Prompt Problems: A New Programming Exercise for the Generative AI Era. In *Proceedings of the 55th ACM Technical Symposium on Computer Science Education V. 1 (SIGCSE 2024)*. Association for Computing Machinery, New York, NY, USA, 296–302. https://doi.org/10.1145/3626252.3630909

[16] Michael Desmond and Michelle Brachman. 2024. Exploring prompt engineering practices in the enterprise. *arXiv [cs.HC]* (March 2024). arXiv:2403.08950 [cs.HC] http://arxiv.org/abs/2403.08950

[17] Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, et al. 2024. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783* (2024).

[18] Anna Eckerdal, Robert McCartney, Jan Erik Moström, Mark Ratcliffe, and Carol Zander. 2006. Can graduating students design software systems? *SIGCSE Bull.* 38, 1 (March 2006), 403–407. https://doi.org/10.1145/1124706.1121468

[19] Molly Q Feldman and Carolyn Jane Anderson. 2024. Non-expert programmers in the generative AI future. In *Proceedings of the 3rd Annual Meeting of the Symposium on Human-Computer Interaction for Work*. ACM, New York, NY, USA. https://doi.org/10.1145/3663384.3663393

[20] FlowGPT. [n. d.]. FlowGPT - The Ultimate Library of ChatGPT Prompts. https://flowgpt.com/. Accessed: 2024-9-2.

[21] freeCodeCamp.org. 2023. Prompt Engineering Tutorial – Master ChatGPT and LLM Responses. https://www.youtube.com/watch?v=_ZvnD73m40o

[22] Vahid Garousi, Sara Bauer, and Michael Felderer. 2020. NLP-assisted software testing: A systematic mapping of the literature. *Inf. Softw. Technol.* 126, 106321 (Oct. 2020), 106321. https://doi.org/10.1016/j.infsof.2020.106321

[23] Qianyu He, Jie Zeng, Qianxi He, Jiaqing Liang, and Yanghua Xiao. 2024. From Complex to Simple: Enhancing Multi-Constraint Complex Instruction Following Ability of Large Language Models. *arXiv preprint arXiv:2404.15846* (2024).

[24] Ellen Jiang, Edwin Toh, Alejandra Molina, Kristen Olson, Claire Kayacik, Aaron Donsbach, Carrie J Cai, and Michael Terry. 2022. Discovering the syntax and strategies of natural language programming with generative language models. In *Proceedings of the 2022 CHI Conference on Human Factors in Computing Systems*. 1–19.

[25] Yuxin Jiang, Yufei Wang, Xingshan Zeng, Wanjun Zhong, Liangyou Li, Fei Mi, Lifeng Shang, Xin Jiang, Qun Liu, and Wei Wang. 2023. FollowBench: A Multi-level Fine-grained Constraints Following Benchmark for Large Language Models. *arXiv [cs.CL]* (Oct. 2023). arXiv:2310.20410 [cs.CL] http://arxiv.org/abs/2310.20410

[26] Zhengbao Jiang, Frank F Xu, Jun Araki, and Graham Neubig. 2020. How can we know what language models know? *Transactions of the Association for Computational Linguistics* 8 (2020), 423–438.

[27] Carlos E Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik R Narasimhan. 2024. SWE-bench: Can Language Models Resolve Real-world Github Issues?. In *The Twelfth International Conference on Learning Representations*. https://openreview.net/forum?id=VTF8yNQM66

[28] René Just, Darioush Jalali, Laura Inozemtseva, Michael D Ernst, Reid Holmes, and Gordon Fraser. 2014. Are mutants a valid substitute for real faults in software testing?. In *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering (FSE 2014)*. Association for Computing Machinery, New York, NY, USA, 654–665. https://doi.org/10.1145/2635868.2635929

[29] Omar Khattab, Arnav Singhvi, Paridhi Maheshwari, Zhiyuan Zhang, Keshav Santhanam, Sri Vardhamanan, Saiful Haq, Ashutosh Sharma, Thomas T. Joshi, Hanna Moazam, Heather Miller, Matei Zaharia, and Christopher Potts. 2023. DSPy: Compiling Declarative Language Model Calls into Self-Improving Pipelines. *arXiv preprint arXiv:2310.03714* (2023).

[30] Tae Soo Kim, Yoonjoo Lee, Jamin Shin, Young-Ho Kim, and Juho Kim. 2023. EvalLM: Interactive Evaluation of Large Language Model Prompts on User-Defined Criteria. *arXiv [cs.HC]* (Sept. 2023). arXiv:2309.13633 [cs.HC] http://arxiv.org/abs/2309.13633

[31] Amy J Ko, Robin Abraham, Laura Beckwith, Alan Blackwell, Margaret Burnett, Martin Erwig, Chris Scaffidi, Joseph Lawrance, Henry Lieberman, Brad Myers, Mary Beth Rosson, Gregg Rothermel, Mary Shaw, and Susan Wiedenbeck. 2011. The state of the art in end-user software engineering. *ACM Comput. Surv.* 43, 3 (April 2011), 1–44. https://doi.org/10.1145/1922649.1922658

[32] Kenneth R Koedinger, Julie L Booth, and David Klahr. 2013. Instructional Complexity and the Science to Constrain It. *Science* 342, 6161 (2013), 935–937. https://doi.org/10.1126/science.1238056 arXiv:https://www.science.org/doi/pdf/10.1126/science.1238056

[33] Terry K Koo and Mae Y Li. 2016. A guideline of selecting and reporting intraclass correlation coefficients for reliability research. *J. Chiropr. Med.* 15, 2 (June 2016), 155–163. https://doi.org/10.1016/j.jcm.2016.02.012

[34] Shuvendu K Lahiri, Aaditya Naik, Georgios Sakkas, Piali Choudhury, Curtis von Veh, Madanlal Musuvathi, Jeevana Priya Inala, Chenglong Wang, and Jianfeng Gao. 2022. Interactive code generation via test-driven user-intent formalization. *ArXiv* (2022). https://doi.org/10.48550/ARXIV.2208.05950 arXiv:2208.05950

[35] A van Lamsweerde. 2009. *Requirements engineering: from system goals to UML models to software specifications*. John Wiley & Sons, Ltd.

[36] 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*. 1–31.

[37] 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 (CHI '23, Article 598)*. Association for Computing Machinery, New York, NY, USA, 1–31. https://doi.org/10.1145/3544548.3580817

[38] Pengfei Liu, Weizhe Yuan, Jinlan Fu, Zhengbao Jiang, Hiroaki Hayashi, and Graham Neubig. 2021. Pre-train, Prompt, and Predict: A Systematic Survey of Prompting Methods in Natural Language Processing. *arXiv [cs.CL]* (July 2021). arXiv:2107.13586 [cs.CL] http://arxiv.org/abs/2107.13586

[39] Qianou Ma, Hua Shen, Kenneth Koedinger, and Sherry Tongshuang Wu. 2024. How to teach programming in the AI era? Using LLMs as a teachable agent for debugging. In *25th International Conference on Artificial Intelligence in Education (AIED) (Lecture notes in computer science)*. Springer Nature Switzerland, Cham, 265–279. https://doi.org/10.1007/978-3-031-64302-6_19

[40] Ggaliwango Marvin, Nakayiza Hellen, Daudi Jjingo, and Joyce Nakatumba-Nabende. 2024. Prompt Engineering in Large Language Models. In *Algorithms for Intelligent Systems*. Springer Nature Singapore, Singapore, 387–402. https://doi.org/10.1007/978-981-99-7962-2_30

[41] Bertalan Meskó. 2023. Prompt engineering as an important emerging skill for medical professionals: Tutorial. *J. Med. Internet Res.* 25, 1 (Oct. 2023), e50638. https://doi.org/10.2196/50638

[42] Lynette I Millett, Martyn Thomas, and Daniel Jackson. 2007. *Software for dependable systems: Sufficient evidence?* National Academies Press.

[43] Lloyd Montgomery, Davide Fucci, Abir Bouraffa, Lisa Scholz, and Walid Maalej. 2022. Empirical research on requirements quality: a systematic mapping study. *Requir Eng* 27, 2 (Feb. 2022), 183–209. https://doi.org/10.1007/s00766-021-00367-z

[44] Daye Nam, Andrew Macvean, Vincent Hellendoorn, Bogdan Vasilescu, and Brad Myers. 2023. In-IDE Generation-based Information Support with a Large Language Model. *arXiv [cs.SE]* (July 2023). arXiv:2307.08177 [cs.SE] http://arxiv.org/abs/2307.08177

[45] Vicente Lustosa Neto, Roberta Coelho, Larissa Leite, Dalton S Guerrero, and Andrea P Mendonça. 2013. POPT: A Problem-Oriented Programming and Testing approach for novice students. In *2013 35th International Conference on Software Engineering (ICSE).* IEEE, 1099–1108. https://doi.org/10.1109/ICSE.2013.6606660

[46] Sydney Nguyen, Hannah Mclean Babe, Yangtian Zi, Arjun Guha, Carolyn Jane Anderson, and Molly Q Feldman. 2024. How Beginning Programmers and Code LLMs (Mis)read Each Other. In *Proceedings of the CHI Conference on Human Factors in Computing Systems (CHI '24, Article 651).* Association for Computing Machinery, New York, NY, USA, 1–26. https://doi.org/10.1145/3613904.3642706

[47] OpenAI. 2023. GPT-4 Technical Report. (2023).

[48] Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. 2022. Training language models to follow instructions with human feedback. *Advances in neural information processing systems* 35 (2022), 27730–27744.

[49] Shuyin Ouyang, Jie M Zhang, Mark Harman, and Meng Wang. 2023. LLM is Like a Box of Chocolates: the Non-determinism of ChatGPT in Code Generation. *arXiv preprint arXiv:2308.02828* (2023).

[50] Savvas Petridis, Benjamin D Wedin, James Wexler, Mahima Pushkarna, Aaron Donsbach, Nitesh Goyal, Carrie J Cai, and Michael Terry. 2024. Constitution-maker: Interactively critiquing large language models by converting feedback into principles. In *Proceedings of the 29th International Conference on Intelligent User Interfaces.* 853–868.

[51] Cheng Qian, Bingxiang He, Zhong Zhuang, Jia Deng, Yujia Qin, Xin Cong, Zhong Zhang, Jie Zhou, Yankai Lin, Zhiyuan Liu, and Maosong Sun. 2024. Tell Me More! Towards Implicit User Intention Understanding of Language Model Driven Agents. *arXiv [cs.CL]* (Feb. 2024). arXiv:2402.09205 [cs.CL] http://arxiv.org/abs/2402.09205

[52] Cheng Qian, Xinran Zhao, and Sherry Tongshuang Wu. 2023. "merge conflicts!" exploring the impacts of external distractors to parametric knowledge graphs. *arXiv [cs.CL]* (Sept. 2023). arXiv:2309.08594 [cs.CL] http://arxiv.org/abs/2309.08594

[53] Yiwei Qin, Kaiqiang Song, Yebowen Hu, Wenlin Yao, Sangwoo Cho, Xiaoyang Wang, Xuansheng Wu, Fei Liu, Pengfei Liu, and Dong Yu. 2024. InFoBench: Evaluating Instruction Following Ability in Large Language Models. *arXiv [cs.CL]* (Jan. 2024). arXiv:2401.03601 [cs.CL] http://arxiv.org/abs/2401.03601

[54] Alex Radermacher, Gursimran Walia, and Dean Knudson. 2014. Investigating the skill gap between graduating students and industry expectations. In *Companion Proceedings of the 36th International Conference on Software Engineering (ICSE Companion 2014).* Association for Computing Machinery, New York, NY, USA, 291–300. https://doi.org/10.1145/2591062.2591159

[55] Mohammad Raza, Sumit Gulwani, and Natasa Milic-Frayling. 2015. Compositional program synthesis from natural language and examples. In *Proceedings of the 24th International Conference on Artificial Intelligence (IJCAI'15).* AAAI Press, 792–800. https://doi.org/10.5555/2832249.2832359

[56] Jekaterina Rogaten, Bart Rienties, Rhona Sharpe, Simon Cross, Denise Whitelock, Simon Lygo-Baker, and Allison Littlejohn. 2019. Reviewing affective, behavioural and cognitive learning gains in higher education. *Assessment & Evaluation in Higher Education* 44, 3 (April 2019), 321–337. https://doi.org/10.1080/02602938.2018.1504277

[57] Douglas C Schmidt, Jesse Spencer-Smith, Quchen Fu, and Jules White. 2024. Towards a catalog of prompt patterns to enhance the discipline of prompt engineering. *ACM SIGAda Ada Lett.* 43, 2 (June 2024), 43–51. https://doi.org/10.1145/3672359.3672364

[58] Inbal Shani. 2023. Survey reveals AI's impact on the developer experience. https://github.blog/news-insights/research/survey-reveals-ais-impact-on-the-developer-experience/. Accessed: 2024-9-2.

[59] Shreya Shankar, Haotian Li, Parth Asawa, Madelon Hulsebos, Yiming Lin, J D Zamfirescu-Pereira, Harrison Chase, Will Fu-Hinthorn, Aditya G Parameswaran, and Eugene Wu. 2024. SPADE: Synthesizing Data Quality Assertions for Large Language Model Pipelines. *arXiv [cs.DB]* (Jan. 2024). arXiv:2401.03038 [cs.DB] http://arxiv.org/abs/2401.03038

[60] Shreya Shankar, J D Zamfirescu-Pereira, Björn Hartmann, Aditya G Parameswaran, and Ian Arawjo. 2024. Who Validates the Validators? Aligning LLM-Assisted Evaluation of LLM Outputs with Human Preferences. *arXiv [cs.HC]* (April 2024). arXiv:2404.12272 [cs.HC] http://arxiv.org/abs/2404.12272

[61] Taylor Shin, Yasaman Razeghi, Robert L Logan IV, Eric Wallace, and Sameer Singh. 2020. Autoprompt: Eliciting knowledge from language models with automatically generated prompts. *arXiv preprint arXiv:2010.15980* (2020).

[62] Arnav Singhvi, Manish Shetty, Shangyin Tan, Christopher Potts, Koushik Sen, Matei Zaharia, and Omar Khattab. 2023. DSPy Assertions: Computational Constraints for Self-Refining Language Model Pipelines. *arXiv [cs.CL]* (Dec. 2023). arXiv:2312.13382 [cs.CL] http://arxiv.org/abs/2312.13382

[63] Dongxun Su, Yanjie Zhao, Xinyi Hou, Shenao Wang, and Haoyu Wang. 2024. Gpt store mining and analysis. *arXiv preprint arXiv:2405.10210* (2024).

[64] Gursimran Singh Walia and Jeffrey C Carver. 2009. A systematic literature review to identify and classify software requirement errors. *Information and Software Technology* 51, 7 (July 2009), 1087–1109. https://doi.org/10.1016/j.infsof.2009.01.004

[65] Qiaosi Wang, Sarah Walsh, Mei Si, Jeffrey Kephart, Justin D. Weisz, and Ashok K. Goel. 2024. Theory of Mind in Human-AI Interaction. In *Extended Abstracts of the 2024 CHI Conference on Human Factors in Computing Systems (CHI EA '24).* Association for Computing Machinery, New York, NY, USA, Article 493, 6 pages. https://doi.org/10.1145/3613905.3636308

[66] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. 2022. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems* 35 (2022), 24824–24837.

[67] Grant P Wiggins and Jay McTighe. 2005. *Understanding by Design.* ASCD. https://play.google.com/store/books/details?id=N2EfKlyUN4QC

[68] Ben Wodecki. 2024. Devin: AI Software Engineer that Codes Entire Projects from Single Prompt. https://aibusiness.com/automation/ai-software-engineer-devin-codes-entire-projects-from-single-prompt. Accessed: 2024-9-2.

[69] Yue Wu, Yewen Fan, So Yeon Min, Shrimai Prabhumoye, Stephen McAleer, Yonatan Bisk, Ruslan Salakhutdinov, Yuanzhi Li, and Tom Mitchell. 2024. AgentKit: Flow Engineering with Graphs, not Coding. *arXiv [cs.AI]* (April 2024). arXiv:2404.11483 [cs.AI] http://arxiv.org/abs/2404.11483

[70] Chenyang Yang, Rishabh Rustogi, Rachel Brower-Sinning, Grace Lewis, Christian Kaestner, and Tongshuang Wu. 2023. Beyond Testers' Biases: Guiding Model Testing with Knowledge Bases using LLMs. In *Findings of the Association for Computational Linguistics: EMNLP 2023,* Houda Bouamor, Juan Pino, and Kalika Bali (Eds.). Association for Computational Linguistics, Singapore, 13504–13519. https://doi.org/10.18653/v1/2023.findings-emnlp.901

[71] Chenyang Yang, Yuchen Zhuang, Jieyu Zhang, Yu Meng, Yining Hong, Grace A. Lewis, Tongshuang Wu, and Christian Kaestner. 2024. What Is Wrong with My Model? Identifying Systematic Problems with Semantic Data Slicing. In *the 39th IEEE/ACM International Conference on Automated Software Engineering.*

[72] John Yang, Carlos E. Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik Narasimhan, and Ofir Press. 2024. SWE-agent: Agent-Computer Interfaces Enable Automated Software Engineering. (2024). arXiv:2405.15793 [cs.SE]

[73] Pradyumna Ym, Vinod Ganesan, Dinesh Kumar Arumugam, Meghna Gupta, Nischith Shadagopan, Tanay Dixit, Sameer Segal, Pratyush Kumar, Mohit Jain, and Sriram Rajamani. 2023. PwR: Exploring the role of representations in conversational programming. *arXiv [cs.HC]* (Sept. 2023). arXiv:2309.09495 [cs.HC] http://arxiv.org/abs/2309.09495

[74] Yue Yu, Yuchen Zhuang, Jieyu Zhang, Yu Meng, Alexander J Ratner, Ranjay Krishna, Jiaming Shen, and Chao Zhang. 2024. Large language model as attributed training data generator: A tale of diversity and bias. *Advances in Neural Information Processing Systems* 36 (2024).

[75] Weizhe Yuan, Pengfei Liu, and Matthias Gallé. 2024. LLMCRIT: Teaching Large Language Models to Use Criteria. *arXiv [cs.CL]* (March 2024). arXiv:2403.01069 [cs.CL] http://arxiv.org/abs/2403.01069

[76] Mert Yuksekgonul, Federico Bianchi, Joseph Boen, Sheng Liu, Zhi Huang, Carlos Guestrin, and James Zou. 2024. TextGrad: Automatic "Differentiation" via Text. *arXiv [cs.CL]* (June 2024). arXiv:2406.07496 [cs.CL] http://arxiv.org/abs/2406.07496

[77] 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 (CHI '23, Article 437).* Association for Computing Machinery, New York, NY, USA, 1–21. https://doi.org/10.1145/3544548.3581388

[78] Zejun Zhang, Li Zhang, Xin Yuan, Anlan Zhang, Mengwei Xu, and Feng Qian. 2024. A first look at gpt apps: Landscape and vulnerability. *arXiv preprint arXiv:2402.15105* (2024).

[79] L Zhao, W Alhoshan, A Ferrari, K J Letsholo, and others. 2021. Natural language processing for requirements engineering: A systematic mapping study. *ACM Computing* (2021). https://dl.acm.org/doi/abs/10.1145/3444689

[80] Jeffrey Zhou, Tianjian Lu, Swaroop Mishra, Siddhartha Brahma, Sujoy Basu, Yi Luan, Denny Zhou, and Le Hou. 2023. Instruction-following evaluation for Large Language Models. *arXiv [cs.CL]* (Nov. 2023). arXiv:2311.07911 [cs.CL] https://github.com/google-research/