Do LLMs Meet the Needs of Software Tutorial Writers? Opportunities and Design Implications

Avinash Bhat
McGill University
Montréal, Canada
avinash.bhat@mail.mcgill.ca

Disha Shrivastava
Google DeepMind
London, United Kingdom
shrivasd@google.com

Jin L.C. Guo
McGill University
Montréal, Canada
jguo@cs.mcgill.ca

ABSTRACT

Creating software tutorials involves developing accurate code examples and explanatory text that engages and informs the reader. Large Language Models (LLMs) demonstrate a strong capacity to generate both text and code, but their potential to assist tutorial writing is unknown. By interviewing and observing seven experienced writers using OpenAI playground as an exploration environment, we uncover design opportunities for leveraging LLMs in software tutorial writing. Our findings reveal background research, resource creation, and maintaining quality standards as critical areas where LLMs could significantly assist writers. We observe how tutorial writers generated tutorial content while exploring LLMs' capabilities, formulating prompts, verifying LLM outputs, and reflecting on interaction goals and strategies. Our observation highlights that the unpredictability of LLM outputs and unintuitive interface design contributed to skepticism about LLM's utility. Informed by these results, we contribute recommendations for designing LLM-based tutorial writing tools to mitigate usability challenges and harness LLMs' full potential.

CCS CONCEPTS

• **Human-centered computing** → **User studies**; **User interface design**; • **Software and its engineering** → **Software notations and tools**.

KEYWORDS

Writing Support, Software Tutorial Writing, Large Language Models

ACM Reference Format:
Avinash Bhat, Disha Shrivastava, and Jin L.C. Guo . 2024. Do LLMs Meet the Needs of Software Tutorial Writers? Opportunities and Design Implications. In *Designing Interactive Systems Conference (DIS '24), July 1–5, 2024, IT University of Copenhagen, Denmark.* ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3643834.3660692

1 INTRODUCTION

Software tutorials refer to the instructional documentation intended to guide the readers progressively through tasks concerning software features. Due to their accessible and engaging style and task-oriented focus, tutorials are indispensable tools for readers learning a new technology [40]. A study by Aghajani et al. [1] reveals that practitioners perceive tutorials as invaluable for numerous software engineering tasks. Beyond supporting software users, tutorial creation significantly benefits its writers, including professional growth and learning [3, 50]. However, creating software tutorials involves a complex interplay of many challenging aspects, including building sufficient technical background of the target technology, writing the reference programs, identifying relevant code snippets to include in the tutorial, clearly presenting and explaining code snippets, and formatting tutorials as incremental stages to facilitate learning [24, 25, 66]. Addressing these aspects insufficiently can result in misleading or faulty content, resulting in poor tutorial quality [2].

Meanwhile, recent advances in language technology have attracted notable attention for their potential in developing effective writing tools. In particular, Large Language Models (LLMs), machine learning models trained with textual data on massive scales to predict and generate language, are increasingly being used to support various aspects of creative writing such as ideation [34], text generation [12], and draft revision [14]. Nevertheless, their potential in software tutorial writing is unexplored. Tutorial writing involves creating instructional content that is engaging, clear and factually accurate across both code and natural language. When trained with large-scale corpora of both natural language and source code, LLMs can generate content across a broad range of topics and predict text and code across multiple natural and programming languages. Such a capacity makes LLMs a suitable candidate for the tutorial writers' toolkit. Moreover, interacting with LLMs is typically through textual prompts, a paradigm that is versatile while requiring minimal effort to learn.

Despite LLMs' potential to be a capable tool, the actual user experience with LLMs can sometimes be filled with uncertainty and dissatisfaction [35]. Notable issues are inconsistencies in model output [19, 68], lack of trustworthiness [21], questions about content ownership [7], and outdated information [28]. The extent to which these issues impact the utility of supporting the tutorial writing process remains uncertain. The non-deterministic nature and sensitivity to changes in prompts [39] also present a significant challenge for tool designers in creating appropriate interactions that can effectively use their capabilities [13, 69]. Moreover, a smooth integration of LLMs into existing processes and tools of the target tasks is far from intuitive [65]. To provide essential support for software tutorial writers with the LLMs, it is, therefore, essential to carefully examine the needs of tutorial writers, how the capacities of LLMs might meet their needs, and how to align such capacities with writers' existing practices and workflows.

DIS '24, July 1–5, 2024, IT University of Copenhagen, Denmark

ACM ISBN 979-8-4007-0583-0/24/07.
https://doi.org/10.1145/3643834.3660692

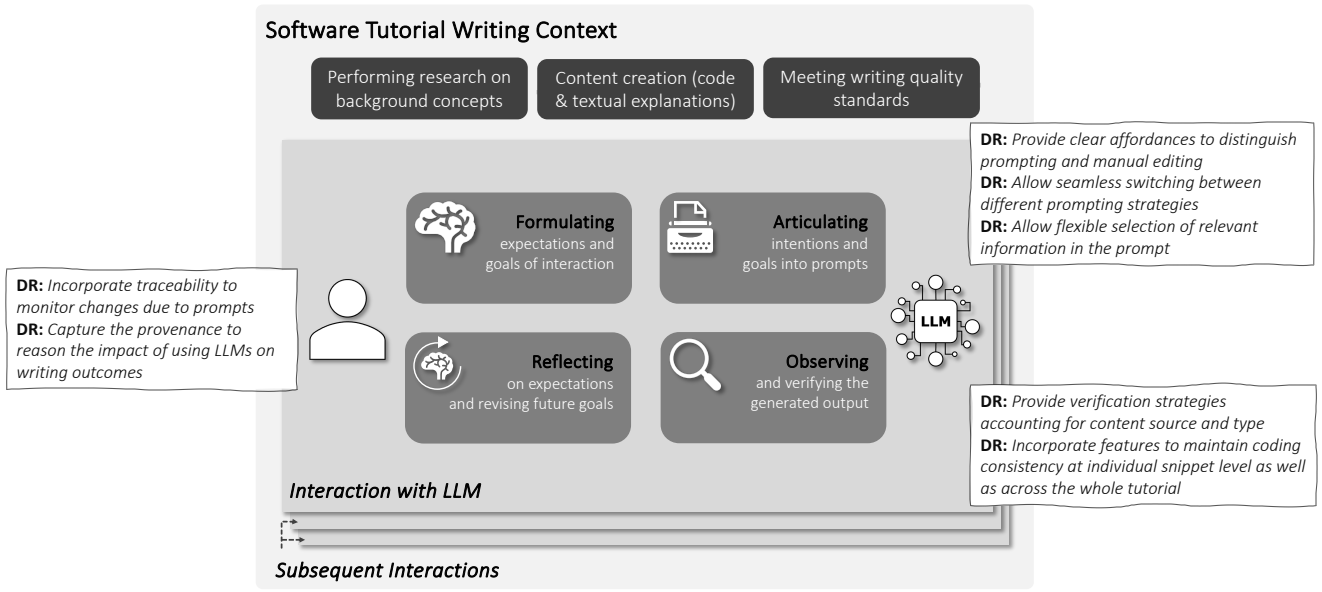

Figure 1: An overview of our key findings, including relevant areas for adopting LLMs in tutorial writing context (derived from interviewing tutorial writers) and aspects during direct interaction (derived from observing writers using LLM for writing tasks). Our proposed Design Recommendations (DRs) for building LLM-based tutorial writing tools are indicated next to the relevant interaction aspects.

In our work, we investigate how LLMs might be effective in meeting the needs of tutorial writers through a user study with seven technical writers with extensive experience in writing and publishing tutorials. Instead of solely focusing on how LLMs can solve tutorial writers' challenges in their existing practice, we also examine novel use cases and interaction patterns of tutorial writers when provided with advanced technology like LLMs. In particular, we started with an interview with the writers to understand their activities and concerns related to tutorial writing and how those activities might benefit from the assistance of LLMs (Research Question 1). Subsequently, we gave them a brief introduction to LLMs' capabilities and limitations and observed their expectations, strategies, and challenges when using an LLM for tutorial writing (Research Question 2). The LLM used in the user study was Codex [47], an LLM specifically trained on both natural language and source code. The interaction was through a web application called playground, offered by OpenAI to enable users to prompt Codex and other models.[1]. We finally discuss our observations to inform potential opportunities and practicalities in designing AI tools for tutorial development. By combining user-centred design with technology-driven inquiries, we contextualize the tutorial writers' values in the expanded innovation space of tutorial tools afforded by LLMs [69, 70].

The interview study results surface three areas that are most relevant to the capabilities of the LLMs in generating code and natural language for tutorial writers: (a) *performing research on background concepts*, (b) *resource creation*, and (c) *meeting writing quality standards*. By observing how writers interact with the LLM, we find four aspects concerning their interaction with LLM-based tools. First, writers approach the interaction process once they *formulate a goal* for the interaction based on certain expectations. These goals involve understanding the model's technical limits or directing the model to produce desired tutorial content. Next, they *articulate their intentions* to the LLM in the form of prompts. Our participants employed strategies to elicit relevant content, such as providing the overall tutorial structure and refining the prompts with topic-specific keywords.

Once the LLM generates an output to the specified prompt, writers *observe and verify* the output in the context of their prompt and the overall tutorial. Verification involves leveraging their domain expertise, consulting existing documentation or references online, and sometimes executing the code generated by the LLM. Finally, writers *reflect and revise* their expectations and future interaction goals based on the usefulness of the output and how well the output meets their expectations. These aspects are performed continuously in subsequent interactions until the objective is achieved or the LLM usage is abandoned. We provide an overview of these stages and aspects in Figure 1.

While resembling the cognitive processes of writing in previous literature [17], the LLM-interaction process we observed is more fine-grained and captures the unique dual objectives of the tutorial authors when using the LLM – understanding the capacity and limitations of the tool and achieving the writing goals. Informed by these findings, we discuss design implications and make recommendations for interface design of LLM-based tools for tutorial writing that can enhance the interaction of users along the highlighted dimensions.

In summary, our work makes the following contributions:

[1]https://platform.openai.com/playground

(1) Identifying unique workflows, considerations, and concerns of software tutorial writing to inform the integration of LLM-based tutorial writing tools.
(2) Depicting the interaction strategies and corresponding challenges faced by the tutorial writers while using LLMs.
(3) Proposing design recommendations for LLM-based tutorial writing tools that address the primary considerations from both the writers' existing workflows and characteristics of their interaction with LLMs.

2 RELATED WORK

Our work is informed by existing research on tool support for authoring software tutorials and interactions for code and natural language generation using LLMs.

2.1 Tool Support for Authoring Software Tutorials

The research landscape for software tutorial creation addresses challenges such as selecting and maintaining consistency in the code examples [20, 24, 25, 66], simplifying the capture of screenshots and other resources to be included in the tutorial [38, 45, 72], and orchestrating capture of crowdsourced information like community annotations for their reader's understanding [15, 22, 32]. The solutions proposed in these studies facilitate integration and maintenance of supporting resources in the tutorial to add context while expecting the writer to manually perform aspects such as developing code implementations, selecting and editing relevant code snippets to include in the tutorial, and crafting high-quality explanations. An interview study by Head et al. [25] highlights the need for support in selecting programming tutorial topics, producing accurate and engaging content, and integrating code snippets with textual explanations. Our work expands this inquiry beyond programming tutorials to include general software technology. We are particularly interested in the applicability and challenges of using LLMs in these workflows – how the software tutorial writer might leverage the LLM's generative capabilities in both code [10, 65] and detailed explanation [36] using natural language statements or prompts [39] to accelerate the tutorial authoring and resource generation process with minimal human intervention. Our work examines user interaction with LLMs for writing tutorials focusing on unique benefits like generating coherent code from natural language and summarizing or explaining code. This approach differs from prior studies, which concentrate on editing tutorials with either reference solutions or expect humans to perform the edits manually.

2.2 Intelligent and Interactive Assistants for Generating Code and Text

While LLMs excel in generating code and natural language, their usability in complex programming and writing tasks is often limited because of the mainstream design of human-LLM interaction. Vaithilingam et al. [65] reported that existing LLM-based tools used for code generation like Copilot[2] generate large blocks of code, making it difficult for humans to debug and refactor code effectively. Barke et al. [4] identified two user interaction patterns while using Copilot: *acceleration*, where programmers use the tool for rapid completion of known tasks, and *exploration*, employed for exploring alternate programming solutions. They used these findings to advocate for better usability of programming assistants, such as providing users with greater control over the code generation and capabilities to validate the generated code. More recently, Ross et al. [54] explored a conversational assistant for general assistance during programming tasks, including code generation, and observed that the conversational paradigm improves the co-creation aspect in code generation. These works highlight the importance of studying human interaction strategies to inform the design of LLM-based tools.

Human-LLM interactions for text generation have been studied across several dimensions, such as needs and values of users [7, 18, 27, 31, 53], writing domains [9, 43, 58], and writing stages [17]. However, existing work on designing tools to support writing activities lacks a discussion on tutorial authoring [17, 33]. For example, by analyzing 33 systems from the literature, Gero et al. [17] map the design space based on the Cognitive Process Theory of Writing [16]. Their work identifies a lack of support in planning and reviewing stages of writing for highly constrained tasks due to the poor capabilities of language technologies at the time. Our work builds upon their result to study the task of tutorial writing in-depth, where we identify the specific writing processes where LLM can be promising. Tutorial writers have open-ended pedagogical goals [30] involving the dual modalities of code and natural language. At the same time, they are tightly constrained by the various aspects of the targeting software, such as the programming language, the underpinning technology, the software version, etc. We investigate strategies and challenges faced by the tutorial writers as they interact with Codex, an LLM capable of generating both code and natural language, aiming to make design recommendations for this open-ended and constrained task. Furthermore, our findings are pertinent to the needs during LLM interactions rather than the general thought process outlined in the Cognitive Process Theory of Writing.

3 STUDY DESIGN

To investigate how tutorial writers interact with an LLM, we conducted an exploratory study with seven highly experienced tutorial writers from diverse backgrounds. Specifically, the goal of the study was to draw out 1) the current workflows and challenges of writers in their tutorial creation process to inform areas where the use of LLMs can be beneficial, 2) the writers' perceptions and expectations when using LLMs, as well as the strategies writers employ to utilize LLMs effectively for their specific needs and expectations. In this section, we discuss the study design to meet our goal. Our study is approved by the research ethics board of the authors' university.

3.1 Participants and Recruitment

We aimed to engage diverse individuals with extensive experience in writing and publishing technical tutorials, ensuring they could provide insights into the challenges, strategies, and opportunities in this area. During the recruitment stage, each potential participant was asked to share at least one of their published technical tutorials;

[2]https://github.com/features/copilot

one author manually validated the tutorials to ensure their expertise. Validation involved checking for sufficient length and depth in the subject matter and the inclusion of instructional resources such as code snippets or screenshots. We had 33 sign-ups for the study from technical writing communities on Slack, Reddit, and LinkedIn, of which 19 were excluded for not sharing links to a published tutorial. Of the 14 who shared the links, four were excluded based on the quality of the tutorial, and three did not proceed with the interview scheduling process. Finally, we recruited seven participants (henceforth referred to as P_1-P_7). Table 1 provides relevant demographic and professional information of all the participants in our study. As a token of appreciation for participating in the study, each participant was compensated with an Amazon gift card valued at $20 CAD or an equivalent amount in their local currency.

Recruiting participants with specialized expertise is difficult to carry out effectively at a large scale. The rigorous recruitment strategy we followed ensures the expertise of the selected participants. Upon inspection, our participants demonstrate sufficient diversity, representing several facets of the software engineering discipline. They provide insights into the documentation practice for open-source communities, startups, and established companies. Their instructional materials are disseminated across multiple platforms, including company websites, community blogging platforms (e.g., Medium), and GitHub. The participant group comprised junior and senior experts in software development and technical writing in terms of years of experience and writing frequency. Given the scope of this work, we deem our study sample is appropriate [23, 46, 61]; input from our participants can provide a rich account of the possibilities and limitations of using LLMs to aid tutorial writers with the generation of software tutorial content and resources in various context.

3.2 User Study Procedure

The study consisted of a semi-structured interview about existing tutorial writing practices and an observation component where we examined how participants used LLMs for tutorial creation. The complete study for each participant lasted around one hour and was screen-recorded.

Semi-structured interview. The initial part of the study involved interviewing participants [56] to understand their current practices and workflows in tutorial writing. We focused on their experiences, tools used, and techniques for writing, organizing, and maintaining tutorials. We asked the participants to contextualize this discussion using (but not limited to) the tutorials they submitted during the recruitment to understand their practices with concrete examples.

Participant Observation. We performed direct observation [41, 51, 56] to get an accurate understanding of the nuanced interactions with the LLMs, especially in the context of writing software tutorials. Since the LLMs were not prevalent in tutorial writing practice during the study period (August and September 2022), any retrospective account would be insufficient to understand the individual contexts in which the users interacted with the LLM. Instead, we asked the participants to mimic the scenario of writing a tutorial on a topic they were familiar with while being assisted by Codex, one of the most capable models trained on both code and natural language at the time of the study. Participants interacted with Codex through the OpenAI playground [48], a web application for easy access to the OpenAI LLMs. The playground presents a large text area along with a panel where the users can choose the LLM settings, notably, mode of interaction (one of Complete, Edit, or Instruct), model from different model families such as Codex[3], maximum length token (default value of 256), which indicates the number of tokens generated by the LLM per request, and temperature (default value 1). Since most of the participants had not used the tool prior to this study, we provided a brief introduction and introduced the playground settings. Participants were free to modify the settings at any point during the exploration. We asked participants to follow the 'think aloud' protocol [29, 60] during the exploration, encouraging them to voice their thoughts, actions, and expectations as they interacted with the tool. The interviewer occasionally asked participants about their actions and impressions of the interaction with the tool. While the study protocol might result in participants behaving differently due to being observed, we wanted to gain rich insights into the participants' thinking process and perspectives as they used the tool, which is difficult to obtain from other study formats [52].

During the observation phase, we chose to leverage a general-purpose model like Codex over fine-tuned tutorial writing models for two reasons. Firstly, given the open-ended nature of tutorial writing, it was uncertain which specific features of the LLMs the writers might engage with. Opting for a fine-tuned model targeted at a particular task could potentially limit our understanding of the broader applications of LLMs in the context of assisting tutorial writers. Additionally, fine-tuning a model without precise direction could lead to premature optimization for specific tasks, which could possibly skew the user's perceptions towards believing that LLMs are only suitable for those particular aspects. Using a general-purpose model like Codex enabled us to study the diverse aspects of tutorial writing, which could be later used to fine-tune the LLMs for specific objectives targeted at the most desired use cases. Secondly, our objective was to investigate the usability aspects of human-LLM interaction in tutorial writing and derive design considerations. Considering this objective, we design our study methods to post minimal constraints on the model itself and to be applicable amidst the advancements in language technologies.

Reflection. We concluded the study by asking the participants to reflect on their interactions with the LLM for writing tutorials, including its perceived usefulness, advantages or challenges, or any other relevant aspects. We also asked the participants about potential features they expected to have for an LLM-based tutorial writing tool.

3.3 Data Analysis

We performed a qualitative analysis of the audio transcripts of the interview study extracted using Microsoft Teams. We analyzed the participants' reflections about their existing writing practises involving text production or code snippet generation, where LLMs could be leveraged to add value to the writing workflows. In addition, we used screen recordings to observe participants' interactions with the LLM in the playground. Here, we leveraged a hybrid thematic analysis approach to make reflective observations [63]. First,

[3]Codex is discontinued in March 2023 [49].

Table 1: Background of the Study Participants. English proficiency is based on Interagency Language Roundtable Scale [55].

Participant	Years in Software Engineering	Tutorial Authoring Frequency (Past 3 Yrs)	Tutorials Written	Experience with AI tools	Current Occupation	English Proficiency
*P*1	<5 years	Weekly/biweekly	5	Not used previously	Lead, technical documentation	Professional Working
*P*2	<5 years	Once a month	2	VS Code IntelliSense	University Student (Computer Engineering)	Native/Bilingual
*P*3	11-15 years	Once a month	20	Not used previously	Technical Writer	Professional Working
*P*4	11-15 years	2-3 times a week	50	GPT-3 based tools (Jasper AI)	CEO (Technical writing agency)	Native/Bilingual
*P*5	1-2 years	Once in several months	20	VS Code IntelliSense	Student, Technical Writer	Native/Bilingual
*P*6	>15 years	Once a month	50	Not used previously	Technical Writer	Native/Bilingual
*P*7	11-15 years	Once a month	50	VS Code IntelliSense	Software Engineer, Site Reliability Engineer	Full Professional

the first author reviewed the audio transcripts and screen recordings to annotate the salient themes and generate the initial codes. Next, the remaining two authors further critiqued and joined the discussion to ensure robustness. As insights emerged from the interviews, we referred to the literature on writing processes and employed an abductive and retroductive inference [42] strategy. We present our results in the subsequent section and draw parallels to the existing theories on interaction design and discussions regarding tutorial writing.

4 FINDINGS FROM SEMI-STRUCTURED INTERVIEW

From the interview with the tutorial writers, we distill the crucial workflows, considerations and challenges they face to answer: **RQ1: What aspects of the tutorial writing are relevant and might benefit assistance from LLMs?** Specifically, we describe how the writers undertake thorough background research prior to the writing, their development of content along with resources such as code snippets and notes, and the refinement of developed content while adhering to self-imposed quality standards. We also discuss participants' reflections on how LLMs can support them in these workflows after they interact with the models (code-davinci-002 and davinci [49]) available in the playground interface for the writing tasks in our study.

4.1 Assisting with Research of Background Concepts

4.1.1 Existing Practice. Before writing a tutorial, writers perform a thorough research of the existing background information about the topic. Their research typically involves investigating existing resources through various channels, including existing documentation and online platforms like YouTube, internet forums, and Reddit. Through research, they identify gaps in the publicly available content and gauge potential information that the learners might seek regarding the subject. Such a process facilitates their own learning and mastery, especially when dealing with new technology or unique applications of familiar technology. In instances where existing resources do not cover certain information, they leverage their access to developers, if available, for further insights and clarifications.

Participants highlighted two challenges related to the interaction with developers. First, developers often presume that the writers possess a foundational understanding of background concepts during technical discussions (e.g., *"[Developers] expect us to understand certain things in the development area. They don't know that we are totally new to this" [P3]*). This expectation leaves writers, especially those new to the technology, with a difficult task to quickly grasp complex background concepts. The second challenge is when writers are blocked due to developers being unavailable for such discussions (e.g., *"Getting a developer's time is sometimes difficult, especially during the sprint or a deadline" [P6]*). Participants acknowledged that the recent shifts towards remote work had facilitated convenient and productive collaborations, with tools like Slack and Zoom ensuring quicker responses by the developers.

4.1.2 Opportunities. The conventional approach to gathering information for tutorial writing is cumbersome since it involves sifting through scattered documentation or consulting with busy developers. Participants acknowledged the potential of using LLMs to streamline this process. For instance, *P*4 identifies LLMs as a potential stand-in, stating *"It would be like a replacement for a developer to ask technical questions. So, if I can't find a developer then I could ask the model, what does this piece of code, module, or web page do? What is it for?"*. Echoing this sentiment, *P*3 observed, *"Even when not probing developers, we have to get definitions and details from the Internet, for which this is extremely helpful."* While collaborating with developers can be insightful for acquiring information not readily available in the documentation, there is an opportunity to leverage LLMs more effectively in this context. For example, *P*6 suggests enhancing LLMs by training them on the design documentation, *"The design documentation, which is usually internal, often explains the rationales for projects. If you could somehow train them, that'd be valuable to explain the rationales and the intentions"*.

4.2 Creating Instructional Code and Text Content

4.2.1 Existing Practice. Writers create a range of content during the tutorial writing process. Content like illustrative code examples and outputs or screenshots (e.g., a tutorial for using software with a graphical interface) enhance the tutorial's instructional value and, therefore, are developed to integrate into the tutorials. However, they lack sufficient tooling support to tutorial writing specific tasks. While developing programming tutorials, writers first implement and execute the complete code. Having a working implementation adds credibility to the tutorials and facilitates the capture of relevant code snippets and outputs, which offers context and clarity for the readers (e.g., *"You have to rely on [code examples] to make sure [the readers] follow the intentions of the API, and not abuse the API. Then I'll throw in the explanatory content around the logical chunks and explain each chunk and the rationale" [P6]*). Participants reported using IDEs, text editors, and, in one case, even traditional pen and paper to develop the implementations.

When it comes to working with code, they needed to *"involve the support and collaborate with developers as well" [P3]* which required adapting tools like GitHub (e.g., *"Developers are more familiar with the GitHub so it's easier for both of us. But for some technical writers, there will be a learning curve to get used to this" [P3]*). This collaboration tends to benefit the developers as it allows them to identify and rectify previously unidentified issues in the existing documentation (e.g., *"Sometimes the client has never actually done what they're asking before, and it might not actually work the way they think it should. That's what they're looking for partly, somebody to help catch those kinds of issues and errors in their documentation. But that can be very frustrating as a writer" [P4]*).

Writers execute and verify the code implementation in the tutorial at multiple stages, starting from the initial development and again while adding textual explanations. This ensures the final tutorials are free from errors which arise when adding additional content and narratives (e.g., *"I usually make edits to make the content more human readable. It's easy to introduce syntax errors while doing that, so I copy it back to the command line to make sure that I didn't break anything" [P7]*). Participants reported inadequate support towards supplementary tasks like managing references to successful and unsuccessful code implementations or capturing complex screenshots requiring extensive setup (e.g.,*"Software developers have amazing tools because they are software developers. They can make their own tools. Technical writers have unmet needs because we're not software developers for the most part, and we can't make tools, so we have to rely on developers to make tools for us" [P6]*).

Resources like notes or writing templates are often intended as personal reference material to assist the writing process, though they may not be included in the final tutorial. Participants stated that these resources are useful when they encounter challenges like *writer's block*, which poses challenges in coherently articulating ideas (e.g.,*"Writer's block was more frustrating than other [challenges] because I have written something which doesn't make sense, but I don't know what to do" [P5]*). Writers construct notes by documenting their everyday work and recording their solutions to problems they reckon the readers might encounter. Writing templates consist of instructions, checklists, or good practices which are either sourced from public repositories like The Good Docs Project[4] or are based on the writers' own prior experiences in tutorial creation. Participants also reported turning to AI-powered tools such as QuillBot[5], which assist with paraphrasing or restructuring the content to get past writer's block.

4.2.2 Opportunities. The capability of LLMs to generate both text and code can allow writers to avoid constantly switching between various IDEs, text editors, and reference materials. *P6* illustrated the possibility of transforming the traditional tutorial creation workflow, *"It flips the workflow around because instead of first making sure the [tutorial steps] work and then retracing your steps and putting them in [tutorial], here can start with the goal and put something out and then you start to test it out." P7* emphasized the efficiency of this approach, mentioning that what took mere seconds with the LLMs, traditionally *"would have taken easily 30 minutes to put out."* Such a shift in the creation process allows writers to focus more on refining the content and ensuring it connects well with the target audience.

4.3 Meeting Tutorial Quality Standards and the Needs of Readers

4.3.1 Existing Practice. Writers adhere to self-imposed quality standards such as clarity, readability, completeness, and being up-to-date, prioritizing the information needs of their audience while aiming to maintain the tutorial's accuracy and relevance. This focus significantly influences their decisions regarding the tutorial's scope, writing style, and the choice of resources to include in the tutorial.

Given the rapid pace of technological updates, keeping the tutorial up-to-date is a critical challenge in ensuring tutorial quality. Writers either keep track of code changes themselves or rely on developers for updates (e.g., *"Most of the time, [developers] inform us if there are any changes in the code or there is a new release. Sometimes they forget, and when users point out that the tutorial seems obsolete, we update" [P3]*). In cases of minor updates, developers modify the tutorials despite being less experienced in writing, which in turn necessitates further editing (e.g., *"We look for technically strong developers to write tutorials. Then, we find editors who can read their tutorials, clean them up, and improve the writing without breaking the technical accuracy" [P4]*). When significant changes need to be made quickly in fast-evolving fields like machine learning, writers prefer to create new tutorials rather than revisit existing ones (e.g., *"In 8-9 months, there are new versions of tools with new features. You can't go back to your tutorial and change everything. The only way is not to update the tutorial but to write new ones" [P5]*). A proactive strategy discussed by the participants is to design tutorials with a focused scope, covering select features to reduce the extent of necessary updates (e.g., *"A tutorial usually touches lightly on a handful of features, and unless those features change drastically, there's not much maintenance" [P6]*).

Writers greatly value clarity and readability (e.g., *"I edit content to make it more human-readable, pretty, and easily digestible. Like breaking up commands into multiple lines" [P7]*). They aim to

[4]https://thegooddocsproject.dev/
[5]https://quillbot.com/

provide the necessary context within the tutorial to minimize the need for any external references (e.g., *"I don't like sending people to [external] links. I rather synthesize the content, reword it and make it more clear" [P7]*). However, achieving the balance in providing the right amount of details can be challenging since writers need to anticipate and tailor the content based on the reader's technical level (e.g., *"With a very junior-level reader, I might want to include every step but with a senior person, I might jump right to the things that are relevant to them" [P4]*). This balance is crucial in designing tutorials that are not only informative but also instill confidence in users to navigate and explore the system. One strategy is to develop short and focused multi-stage tutorials that gradually increase in complexity and scope. Despite existing tools like Confluence that are used to organize and structure the tutorials, articulating concise tutorials remains challenging (e.g., *"Trying to condense and be to the point but also remain very clear and read well is the challenge" [P7]*).

4.3.2 Opportunities. Participants acknowledged LLMs' efficiency in tasks like translation, which is essential for tutorial content dissemination. Translation extends content accessibility and broadens its reach. Multilingual support is often a requirement posed by the companies, *"In EU, if you have documentation on your site, you have to have it in the native country's language as well. The German and French companies like what we write [in English], but also want to have it in French and German, their native languages, and that's a big deal for them" [P5]*. However, the challenge lies in ensuring that the translated content maintains its technical accuracy and contextual relevance. Versions in different languages need to maintain the same level of accuracy and clarity as the original, often necessitating human oversight (e.g., *"I don't see how any model, even if it works, will just write the things. How will it know to maintain itself?" [P1]*).

5 FINDINGS FROM PARTICIPANT OBSERVATION

To further understand the design considerations for using LLMs in tutorial writing, we must investigate how the writers might approach LLM interaction for concrete writing tasks. In this section, we draw from observations of how participants utilized the LLM in tutorial creation to answer: **RQ2: What are the expectations, strategies, and challenges when writers use LLMs for tutorial creation?** In particular, we discuss how writers formulate initial expectations and goals of interaction, articulate their goal through prompts to LLMs and other parameters, observe and verify the generated content, and eventually reflect and revise their goals and interaction strategies based on the output.

The prospect of using LLMs for writing a complete tutorial was new to all our participants, given the lack of established and mature LLM-based tutorial writing tools in the market; four participants (*P2*, *P4*, *P5*, *P7*) had prior experience with using intelligent coding tools (see Table 1). Therefore, each participant was briefly introduced to the capabilities of individual models (code-davinci-002 and davinci [49]) before the observation study and the features of the OpenAI playground that might be relevant to their writing process. We encouraged the participants to use LLMs for broader tutorial writing workflows that they discussed in the interview, as described in the previous section. Examining their interactions within the context of outlined tasks situates their perceptions, strategies, and challenges in adapting LLMs to the unique requirements of tutorial creation. When introducing the quotes from the participants, we indicate the actual textual prompts our participants typed on the playground interface by enclosing them in brackets and highlighted with a [different font] for clarity.

5.1 Formulating Expectations and Goals of Interaction

Participants possess a mental model of the LLM, i.e., an internal representation of the LLM's functionalities, capabilities, and anticipated behaviour. **Users form expectations and define the goals and intentions according to their mental model** as they approach the LLM in each interaction cycle. The objectives of the interaction might be to update the mental model (e.g., to probe and understand the capability of LLMs better or to explore the possible ways to elicit the best responses from it) or to use LLM towards the overall writing goal (e.g., to complete a section in the tutorial or to edit a specific step in the tutorial). While working with an LLM, the interaction goals frequently switch between the two as the user attempts to decipher the LLM's capabilities and employ them for tutorial writing.

Some participants intended to start by replicating the strategies shown in the initial introduction of the study (e.g., *"Just repeating whatever worked successfully before, as you showed me" [P6]*). As familiarity with LLMs grew, participants began experimenting and calibrating the workflows (e.g., *"Let me use a different input* [Exploring different Embeddings]. *I want to see what comes up" [P5]*). Such experimentation was often related to understanding the capacity of LLM (e.g., *"Can we describe something more sophisticated than just a single function? Say I gave it the source code to an entire program and see if it understands how a human would use it?" [P6]*).

Since tutorials are often written for a particular version of a specific technology, participants expressed the **need to understand the model's technical boundaries and the sources of the information** (e.g., *"Is the model working from the information they got from the help content, like the documentation or source code?" [P6]*). However, such information is not readily available or easily assessed, leading to questions like *"How quickly do they update the model? At one point they indexed a lot of stuff from Google results. Are they doing that continuously?" [P4]*. To obtain this knowledge, participants intend to leverage circuitous strategies (e.g., *P4* described a strategy they employed *"Trying to figure out the limit, I started playing around* [tell me about the latest updates to Redis], *and see which version it tells me about because that gives you the decay of how updated it is"*). Prior experiences and perceptions about the capabilities of LLMs influenced the participants in their probing (e.g., *"I've seen one example on the Internet about a security issue when you would ask [the LLM] about API keys, and it will give you someone's API keys. What happens if you tell the Playground to generate AWS keys?* [generate AWS keys]*" [P1]*).

Participants reflected being skeptical and hesitant since they did not understand the inner workings of the model (e.g., *"We don't understand what this model does. I don't know what I'm entering, and I don't know what's happening with it. I don't wanna use it" [P1]*). While there is interest in **understanding the LLMs' training**

processes, data update cycles, strengths and weaknesses, it is primarily to gauge the tool's relevance and utility for tutorial writing. *P*6's query, *"The model knows about [software] due to its training on documentation. But what about when documentation is absent?"*, pinpoints the desire for clarity on how LLMs acquire and utilize information.

5.2 Articulating Goals and Intentions into Prompts

Users communicate their intentions to the LLM through carefully constructed prompts and sometimes even specific LLM parameters. However, **accurately articulating their intentions can be challenging, leading to ambiguous or misdirected prompts**. The user's perceptions of the capability of the LLM can skew their prompts toward either oversimplification or excessive ambition. Consequently, users found themselves dedicating significant effort towards refining their prompts to effectively *"phrase it for the machine" [P4]*. This task is further complicated by a lack of transparency in how changes in prompts have impacted the LLM's outputs.

As the participants worked to align the LLM's output with their writing objectives, they devised multiple strategies to steer the LLM (e.g., *"Everything that I've considered valid here, I'll retain, and then I'll guide [the model] in a slightly different direction" [P7]*). Their **prompting techniques mirrored traditional manual tutorial drafting approaches**, such as providing an overarching structure of the target tutorial. *P*5, for example, structured their tutorial by starting with section titles, noting, *"I want to start with an introduction. I would probably input the title* [Searching for Semantic Similarity - Introduction] *and see the [LLM's] response"*. Participants also **frequently edited and reformulated their prompts, such as adding/removing topic-specific keywords** (e.g., *"To avoid bias, I removed NLTK [from the context window], prompting it to explore GloVe. When I excluded GloVe and added the word choosing, it began suggesting alternatives. It eventually provided three sensible options" [P5]*). Crafting and tinkering with prompts to achieve the intended output required considerable time and effort from the participants. Experienced participants questioned the actual value derived from using the LLM (e.g. *"It takes this art form to get it to actually produce relevant output. I have to think about what am I actually getting it to do. When I am a developer who's done this for many years, it would be faster for me to do it myself" [P4]*).

The usability of an LLM's interface significantly influences how users understand its capabilities. While we acknowledge that the playground's primary aim is model exploration and not tutorial writing, we observed that **the usability issues led to misconceptions and eventually resulted in underutilization or abandonment of the LLM**. Two primary issues of the interface were the inability to differentiate between prompts and generated text and unclear indicators of how prompts and model parameters affect text generation. The playground provides a single text box for both input and output in the Complete mode, confusing the participants about what constitutes input for the LLM (e.g., *"There's nothing related to deployment [in the generated content] because it's biased by the multitude of input before deployment" [P5]*). This confusion led to strategies like pruning the existing content in the text box to manage context. Nevertheless, participants expressed skepticism about adopting the interface to the actual tutorial writing (e.g., *"If you want to write a blog in continuation, how can you not have the whole next thing? Is it expected to completely remove content every time to let this tool do the job?" [P5]*). Furthermore, the lack of clear indicators to illustrate the effect of prompts or model parameters necessitated the participants to often speculate on the required prompts and parameters to obtain desired output (e.g., *"I'm sure it's not gonna be able to create all that in just 256 characters. I guess I can up [maximum token length] and see what happens" [P4]*). Several participants observed the generation stopping mid-sentence due to a tool-imposed limitation on the token length *"OK, so I ran out of tokens there" [P6]* and resorted to workarounds to continue generation *"I guess if I hit 'submit' again, is it gonna keep going or what? What would it do?" [P4]*. However, such strategies were not apparent and resulted in judgments like *"fall short, but at least complete a sentence" [P5]*. While the playground offers modes like Complete, Edit, and Instruct, their utility was not obvious or cumbersome as the participants had to copy and paste the target content between the modes manually. Participants attributed the low usability of features as one of the biggest factors for abandoning such tools for tutorial writing (*"I think there are too many issues for it to be worth working on it. I have no idea how to make it usable" [P1]*).

Regardless, there is a constant disconnect between user intent and how to prompt the output, as illustrated by *P*5's comment, *"for this [model], you'll have to find what exactly to tell them. It's like communicating in a different language."* Participants also indicated a need for fine-grained control over content generation and editing. As *P*5 put it, *"it should let me provide some basic keywords that I want to include and not just go on its own spree of doing whatever it wants."*

5.3 Observing and Verifying the LLM Generated Output

Users observe the accuracy and relevance of the generated output concerning their initial prompts and the current context of their writing. The accuracy is verified through domain expertise, cross-referencing with reputable external sources, or testing the real-world applicability by executing it. Beyond factual accuracy, the output must also align with the user's intentions, which is ensured by evaluating the generated content against the user's original prompt. Other important aspects include verifying the tone and style of the content and the consistency with the content created from the previous interaction iterations.

Participants drew upon their domain expertise or general knowledge to verify the generated content in certain instances. Participants with prior experience in the concerned technology were able to leverage their knowledge to identify discrepancies in the output (e.g., *P*7 leverages their experience as a developer to identify incorrect content about AWS access keys, *"I think there are some missing steps here. Here's the thing, because we just created the account these access keys will not exist at this point"*). Common world knowledge is also used for tasks like translation (e.g., *"'getting started' doesn't translate to 'à propos de départ.' It doesn't mean anything" [P1]*). When domain knowledge was insufficient, **participants chose**

strategies such as cross-referencing with existing documentation, internet search, or testing by execution. Minor details of generated content (e.g., URLs) were checked for authenticity and correctness by a browser search or against existing documentation. Aspects that involved complex reasoning (e.g., code snippets or steps for creating an AWS account as explored by *P*7, *"let's actually go ahead and test all of this"*) were tested through manual execution.

Participants observed the textual quality of the generated content. As detailed in section 4.3, participants desire specificity and coherency in the generated content (e.g., *"I was hoping that [the generated content] would be a bit more specific when it says 'run the installer.' I mean, it's pretty obvious for the end user, but I like to make it impossible to do the wrong thing" [P7]*). Often, they resorted to manually editing the generated content (e.g., *P*7 mentions *"I would remove extra information [from the generated content]. That's my style"* and proceeded to edit manually) and frequently switched between using the text box to prompt the LLM and manually make changes to reach the final outcome.

Participants felt that human oversight and verification remain indispensable for capturing technical nuances and maintaining accuracy. The trust in generated content is contingent on having the control to verify further and edit, as illustrated by *P*7, *"I would not say that I have enough trust that I can publish it and call it a day. I would probably still want to test it, at least go through the steps."* However, verification has its own challenges, considering the security implications of LLM-generated content, as illustrated by *P*7's comment *"Considering that an algorithm generated this code, maybe we need some kind of sandbox so that it doesn't do any damage to my system"* who later called for such a feature to be implemented in the interface. Considering the instructional aspect of tutorials, participants were particularly cautious as any misinformation could significantly derail the learning experience (e.g., *"When I was asking where the source code for [software] was, it didn't know, and it gave an incorrect answer. It'd be cool if it could just touch its shoulders and say, I don't know, or give some kind of score about how confident it is about its answer" [P6]*).

5.4 Reflecting on Expectations and Revising Future Interaction Goals

Users reflect on their latest interaction to **revise their mental models and calibrate interaction strategies with the LLM** or **update their writing objectives for the tutorial**. Reflection assists the users in comprehending the underlying reasons for any discrepancies between the expected and the actual output and updates the mental model to manage the expectations. Users accept the LLM's output when the discrepancies are minimal. In addition, they positively update their perception of the LLM's capabilities and form an enhanced mental model of the interaction strategies. Conversely, when users frequently encounter significant discrepancies and continuously need to calibrate their strategies, it signals a disconnect between their expectations from the LLM, the prompts, or their reflection process.

We observe that the reflection process is grounded in the user's hypothesis of the functioning of the LLM and how it handles the prompts. For example, *P*4 described the working of LLM as *"It isn't really contextually aware. It's just pulling text and trying to figure out what text makes sense around that text"*). However, participants found it **difficult to form these hypotheses due to the absence of traceability between prompts and their outputs** (e.g., *"I've modified two things in this latest query, so the output is a little different than before. I'm not sure what made it different" [P6]*). Even when formed, these hypotheses are not necessarily accurate (e.g., *"It needs 'machine learning' [as a keyword] in the prompt. It doesn't work for any other thing" [P5]*). Participants adjusted and refined their initial hypotheses when the generated content did not align with their expectations. For example, when the model did not generate any content about the software they mentioned in the prompt, *P*4 mentioned *"I think for [software], it's even more niche. It's not a well-known tool. So the problem is it's probably not much content to pull from. It probably just ignored [software] as a tool"*. Users verified the hypothesis with a subsequent interaction cycle or by referencing external material. For example, *P*6 refers to the software's hosted documentation with *"I'm curious to see how it's actually listed [on the website]. Is the [generated] text verbatim?"*. Eventually, we observed instances where the participants decided to let the LLM take control when they were confident in its capabilities (e.g., *"There's some kind of differences in configuration for different host OS that I know of, but I'll let it figure that out for me" [P7]*).

Reflecting on the LLM-generated content resulted in revisions to the writing objectives. The revisions could be towards expanding the scope (e.g., *"Nice, it even added a 'Clean up' section. This would be good to elaborate on" [P7]*) or towards enhancing the quality (e.g., *"I will actually verify and edit if I find that there's things that are either not working or maybe extra information. Like I mentioned, I try to be to the point" [P7]*). Participants considered the generated content as the first draft, which they could enhance using their experience (e.g., *"I could use this to help me bootstrap and get started and get a basic version going. Then I can fill in the gaps" [P4]*).

6 DISCUSSION

In sections 4 and 5, we described where LLMs are considered most relevant in the tutorial writing process and highlighted the challenges writers encounter while interacting with these novel language technologies. Building on these findings, we discuss three major design implications and provide recommendations for developing LLM-based tutorial writing systems. We summarize these in Table 2. We also examine the potential limitations of our study and propose future research directions in LLM-assisted tutorial writing.

6.1 Implications for Designing LLM-Based Software Tutorial Writing Tools

In this section, we discuss the design implications derived from our findings outlined in Sections 4 and 5. We propose recommendations to enhance the functionality and user experience of LLM-based tutorial writing tools while focusing on the opportunities in the existing tutorial writing practises. The recommendations highlight three key areas: assisting writers in improving their mental models of LLMs, enhancing the control over content generation and editing, and verifying the accuracy and relevance of generated content.

6.1.1 Design Implication #1: Assisting Tutorial Writers in Forming Accurate Mental Models of LLMs. As outlined in Section 5.1, the writers' expectations and approach towards using LLMs for generating

Table 2: Design Implications and corresponding Design Recommendations distilled from the findings.

Design Implications	Evidence	Design Recommendations (DRs)
Section 6.1.1: Writers should be able to develop sufficiently accurate mental models to set realistic expectations from the LLM and to carry out appropriate evaluation of outcomes relevant to the tutorial.	**Sections:** 5.1, 5.4	**DR 1.1:** Incorporate traceability to map code and natural language changes between prompt-output pairs within and across interaction cycles. **DR 1.2:** Capture the provenance of the manually written and LLM-generated content to better reason the impact of using LLMs on their writing outcomes.
Section 6.1.2: Writers should possess clear prompting mechanisms and be able to control tutorial-specific content generation and editing precisely.	**Sections:** 4.1, 4.2, 4.3, 5.2	**DR 2.1:** Separate and clearly distinguish the interface between prompting LLM and manually inputting text. **DR 2.2:** Provide a flexible interface that allows seamless switching between diegetic and non-diegetic prompting and editing that happen at different stages of tutorial writing. **DR 2.3:** Allow writers to incorporate and flexibly select relevant sections of the current tutorial draft and prior background research to perform targeted edits of source code or explanations.
Section 6.1.3: Writers should be equipped with diverse verification features to ensure the reliability and relevance of tutorial content generated by LLMs.	**Sections:** 4.1, 4.2, 4.3, 5.3	**DR 3.1:** Provide tailored verification mechanisms based on the source (e.g., incorporating research notes or reference code) and type (e.g., code interpreters for code snippets and link checkers for URLs) of the content type used in the tutorial. **DR 3.2:** Provide features such as static analysis tools and continuous integration systems to ensure consistency in syntax and conventions within individual code snippets and across the combination of all snippets throughout the tutorial.

tutorials are shaped by factors such as their background research on the tutorial topic, personal impressions of LLMs formed from previous experiences, their curiosity to explore LLMs' capabilities, and their progress in the current LLM interaction cycles. In Section 5.4, we further described how the writers reflect their interaction objectives, specified prompts, and the corresponding outputs. The reflection process contributes to changes in their mental model of the LLM and leads to adjustments in their strategies to use the LLM for their writing tasks. However, the mental model is often nascent, lacking a comprehensive understanding of the underlying technology, such as training and fine-tuning of the LLMs, the meaning and impact of parameters like temperature, as well as effective prompting strategies [8, 67]. Prior research has observed an existence of *gulf of envisioning* resulting from insufficient knowledge of LLM's capabilities [62]. In our study, the participants expressed the need to understand the model's technical boundaries and the training data cutoff date to ensure the predictions are accurate and correspond to the latest software versions (Section 4.3). Ensuring better transparency of the underlying models using model cards [44] or explainable AI techniques [57] can be useful to address this issue.

However, expecting end users, tutorial writers in our case, to acquire the relevant LLM-related knowledge may not be practical, as they might not possess the required technical expertise or simply may lack the interest to do so (e.g., *"I still need to double-check everything and figure out how to make it actually generate things you want. Now I have to learn machine learning properly and keep up to date with all these things. And it's just, well, maybe I'm not going to do that" [P1]*). The writers need an effective (not necessarily comprehensive) mental model that can help reason the impact of their actions and what to do next. Incorporating **traceability to map code and natural language changes across LLM interactions** and **provenance to record the tutorial's evolution** can be two actionable design options that can offer two benefits. First, it assists the writers in forming effective mental models through self-evaluation, which is hypothesized to improve metacognition in GenAI systems [64]. Tracing the edit history over a series of interactions helps writers better understand the effect of individual prompts over content evolution and assists writers in calibrating their expectations.

Second, software tutorial writing is often open-ended but constrained by coherence across two modalities, i.e., code and natural language. Design features that support traceability and provenance can be especially useful in generating both modalities when the writers lack a clear prompting strategy and evaluation roadmap. Previous work has demonstrated that traceability between the code snippets and corresponding documentation attribute to improved documentation practises [6], and consistency and accuracy of the

overall tutorial [25]. When part of the code and its natural language explanations of a tutorial are generated by an LLM, traceability provides additional value to track consistency between the human-written and AI-generated content between code and natural language and across several interaction cycles, thereby improving the tutorial's accuracy.

6.1.2 Design Implication #2: Improving Control over Technical Content Generation and Editing. Writers struggle to formulate prompts that accurately reflect their content generation objectives and generalize their prompting strategies across various scenarios [71]. Such issues lead to the perception of the LLM being unpredictable. Furthermore, the OpenAI playground lacks clear signifiers for prompting and observing the generated content, thereby adding to the confusion. As outlined in Section 5.2, using a single text area for the dual purpose of prompting the LLM to generate content as well as manually editing tutorial content resulted in the writers struggling to differentiate between the existing tutorial content, the new prompts, and their outputs. To find a workaround, writers formed complex yet mostly unsuccessful strategies to guide the model, often leading to more frustration.

Given these challenges, it is crucial to explore alternative strategies and interactions that could enhance the control writers have over tutorial content generation. Previous research by Dang et al. [11] investigated two prompting interactions – diegetic (i.e., narrative style and current writing context are part of the prompt) and non-diegetic (i.e., prompts are *not* part of the narrative and are instruction-oriented). Their work suggests diegetic and non-diegetic interactions as effective for inspiration and control, respectively. We observe the benefits of both interactions in different contexts. Diegetic interactions can be useful for tutorial writers in leveraging writing templates as a part of the current writing context (Section 4.2) and expecting the LLM to generate the tutorial according to the template. On the other hand, writers edit the content to meet their standards of tutorial quality. In this case, non-diegetic prompting can enable the writers to get suitable suggestions based on their researched material (Section 4.1) or refine specific sections to meet quality standards (Section 4.3). For instruction-oriented prompting, **separating input fields for LLM prompts and manual text edits** is extremely important to distinguish user-provided context from tutorial content. Furthermore, writers apply a combination of these prompting strategies and even resort to manual edits when they consider LLMs too complex to be used. To account for these editing strategies, the system must offer a **flexible interface that allows seamless switching between diegetic and non-diegetic prompting and editing**, allowing writers to leverage LLMs across varying contexts.

Understanding the tutorial writing practice offers important hints to improve the writers' control further when interacting with LLMs. Tutorial writing is not a linear process but rather involves multiple rounds of investigation before and during the tutorial writing process to develop a better understanding of the target technology (Section 4.1). In addition, writers include resources such as code examples, their outputs, screenshots, and explanations to improve the tutorial's comprehensibility and credibility, as detailed in Section 4.2. Allowing LLMs to reference relevant material from the writer's investigation, such as the existing tutorial on the same technology, can facilitate prompting for detailed, topic-specific content. Therefore, **enabling users to incorporate and flexibly select specific sections of existing material as prompt context** can be useful to create the right prompts.

6.1.3 Design Implication #3: Facilitating Verification to Ensure Tutorial's Accuracy. Participants recognized the potential of LLMs to assist in researching existing resources and even viewed LLMs as potential stand-ins for developers (Section 4.1). However, they expressed reservations about directly using content generated by LLMs in their tutorials and insisted on verifying the content (Section 5.3). The hesitation arises because writers traditionally rely on resources authored by other human writers, allowing for a clear judgement of their credibility implicating a sense of control [7]. Qualities like accuracy, clarity and being up-to-date are crucial to software tutorials (Section 4.3). Therefore, having a verifiable source is critical to the writers' confidence when synthesizing existing resources and creating new content. Despite the advancements in novel fine-tuning [26] and prompt engineering techniques [37], which enable LLMs to contextualize and generate more relevant information, the tangible benefit of manual verification can be indispensable for fostering confidence in the resulting content. Our participants tend to verify the output against their personal expertise or knowledge (Section 5.3). However, human memory is often unreliable, subject to *evaluator fatigue* [5] and complex metacognitive demands [64], which is suboptimal for evaluating tutorials that demand a rigorous quality.

One way to support writers in ensuring the accuracy of LLM-generated content is to **provide tailored verification strategies relevant to the source and type of each content type** used in the tutorial. Writers typically verify the LLM-generated content by cross-referencing it with trusted external sources, leveraging their domain knowledge, or conducting tests to assess real-world applicability (Section 5.3). Traditionally, writers create and reference material, such as code examples and notes (Section 4.2), to improve the accuracy and credibility of the tutorial. Allowing access to these resources within the interface has the potential advantage of minimizing the need for external fact-checking, thereby reducing the associated cognitive load. Verification techniques can differ depending on the type of content as well. For instance, verifying URLs requires confirming their online existence and ensuring the relevance and accuracy of the information on the linked page. In addition, the description of the web resource in the LLM-generated content should accurately reflect the source material. Unlike URLs, verifying code snippets involves providing a code interpreter into the interface to execute and ensure accuracy.

A challenge with using LLMs for tutorial writing is maintaining consistent syntax and conventions within individual code snippets, as well as functional integration across all the snippets in the tutorial. Software tutorials often include several code snippets, which may be parts of an individual program interspersed with explanatory text (e.g., an introductory Python tutorial explaining function calls and loops) or a combination of multiple programs (e.g., a tutorial on setting up a website using a web framework and a database). In both scenarios, ensuring consistency involves two critical steps. First, each snippet must be checked for consistency. Second, the

snippets should collectively form a coherent and executable program that aligns with the provided explanations. **Implementing static analysis tools and continuous integration systems triggered at every revision of the tutorial content** can be useful to maintain consistency and enhance the overall quality and reliability of the tutorial.

6.2 Limitations and Future Work

While our study provides valuable insights into using LLMs for tutorial writing, we acknowledge certain limitations that highlight areas for future work. The first limitation concerns unrealistic expectations or insufficient AI knowledge among users in user-centred design processes, which results in poorly built prototypes that fail to deliver value [69, 70]. While our participants indeed lacked an in-depth understanding of the recent AI technologies, the observations and perspectives shared by the tutorial writers while using the LLM helped us understand their mental models of such technology and its capabilities, thereby informing their novel needs when interacting with AI. Yildirim et al. [70] advocate for a similar approach to enhance user-centred design for AI tools, suggesting that demonstrating AI's capabilities and limitations can guide user interactions more effectively. They recommend leveraging AI experts to validate the practicality of designs arising from the study before development. Correspondingly, our findings emphasize the importance of collaborative efforts between LLM developers and system designers in the user-centred design process. Users might be unable to clearly identify and state their requirements or needs. In such situations, understanding both their existing practises and how they navigate a new technology can help us identify the gap in their understanding. Our work further hints at the importance of scaffolding users' exploration of technology boundaries and innovations, given the inexhaustible possibilities of novel use of LLMs. In our study, we found that while the writers lacked specific knowledge about using AI for tutorial writing, they actively explored and adapted to these gaps. This adaptability might stem from their professional role as tutorial writers, which requires experimentation with software to develop functional tutorials. While our design recommendation of incorporating traceability and provenance is useful to support such user-initialized experimentation, future work is needed to investigate the effectiveness of our recommendation and other means.

Second, the study investigates the writers' experiences using the GPT-3.5 (code-davinci-002 and davinci [49]) models available within the playground interface. Using models fine-tuned for tutorial writing or interfaces tailored specifically for tutorial writing might result in different experiences for writers than we presented in the study. However, as far as we know, such interfaces do not currently exist. The design recommendations discussed in Section 6.1 are intended to guide the development of such novel interfaces with specialized functionalities that can assist writers. Better tutorial writing experience might also benefit from developing models fine-tuned on software tutorials, prompt engineering techniques like introducing code repository-specific knowledge during prompt design [59] and designing corresponding affordances.

The third limitation is that the writers' strategies and perceptions evolve as they interact with the LLM. The study results have implications for the usability and learnability of the system in the short term, contributing to its adoption. However, these results do not inform any change in perception resulting from long-term use. Additionally, the study could not account for the influence of external factors on participants' perceptions, such as public opinion on LLMs. A longitudinal study examining the evolution of writers' interaction strategies and perceptions over extended periods of LLM use could further confirm the integration of LLMs in tutorial writing processes.

7 CONCLUSION

In this paper, we describe the opportunities and challenges for LLMs in tutorial writing. Our findings are based on a user study of software tutorial writers with diverse backgrounds and expertise using the OpenAI playground as an exploration environment. From the interviews, we identified *performing background research*, *resource creation*, and *meeting writing quality standards* as the three areas that are especially relevant for LLM adoption. We observed how the writers *formulate goals*, *articulate* intentions, *observe* and verify outputs, and *reflect* to revise the subsequent strategies when interacting with the LLM. Based on our findings, we surface three design implications, which include 1) effective affordances to develop accurate mental models, 2) clear prompting mechanisms and control over tutorial content generation and editing, and 3) diverse verification features that account for content type and self-created resources. These implications hold potential for better interface design for LLM-based tutorial creation tools and tools that support the generation of both natural language and code.

ACKNOWLEDGMENTS

We thank our anonymous participants for their valuable insights, and the DIS reviewers and ACs for their constructive feedback on the manuscript. We thank Deeksha Arya for her support and invaluable feedback during the qualitative coding process. Avinash Bhat and Jin L.C. Guo are supported by the Natural Sciences and Engineering Research Council of Canada (NSERC).

REFERENCES

[1] Emad Aghajani, Csaba Nagy, Mario Linares-Vásquez, Laura Moreno, Gabriele Bavota, Michele Lanza, and David C. Shepherd. 2020. Software documentation: the practitioners' perspective. In *Proceedings of the 42nd International Conference on Software Engineering (ICSE)*. ACM, New York, NY, USA, 590–601. https://doi.org/10.1145/3377811.3380405

[2] Emad Aghajani, Csaba Nagy, Olga Lucero Vega-Márquez, Mario Linares-Vásquez, Laura Moreno, Gabriele Bavota, and Michele Lanza. 2019. Software documentation issues unveiled. In *Proceedings of the 41st International Conference on Software Engineering (ICSE)*. IEEE / ACM, 1199–1210. https://doi.org/10.1109/ICSE.2019.00122

[3] Deeksha M. Arya, Jin L. C. Guo, and Martin P. Robillard. 2024. Why people contribute software documentation? In *Proceedings of the 17th International Conference on Cooperative and Human Aspects of Software Engineering (CHASE)*. https://doi.org/10.1145/3641822.3641881

[4] Shraddha Barke, Michael B. James, and Nadia Polikarpova. 2023. Grounded Copilot: How Programmers Interact with Code-Generating Models. *Proceedings of the ACM on Programming Languages* 7, OOPSLA1 (2023), 85–111. https://doi.org/10.1145/3586030

[5] Advait Bhat, Saaket Agashe, Parth Oberoi, Niharika Mohile, Ravi Jangir, and Anirudha Joshi. 2023. Interacting with Next-Phrase Suggestions: How Suggestion Systems Aid and Influence the Cognitive Processes of Writing. In *Proceedings of the 28th International Conference on Intelligent User Interfaces (IUI '23)*. ACM, New York, NY, USA, 436–452. https://doi.org/10.1145/3581641.3584060

[6] Avinash Bhat, Austin Coursey, Grace Hu, Sixian Li, Nadia Nahar, Shurui Zhou, Christian Kästner, and Jin L.C. Guo. 2023. Aspirations and Practice of ML Model

Documentation: Moving the Needle with Nudging and Traceability. In *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems (CHI '23)*. ACM, New York, NY, USA, 17 pages. https://doi.org/10.1145/3544548.3581518

[7] Oloff C. Biermann, Ning F. Ma, and Dongwook Yoon. 2022. From Tool to Companion: Storywriters Want AI Writers to Respect Their Personal Values and Writing Strategies. In *Proceedings of the 2022 ACM Designing Interactive Systems Conference (DIS '22)*. ACM, New York, NY, USA, 1209–1227. https://doi.org/10.1145/3532106.3533506

[8] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel Ziegler, Jeffrey Wu, Clemens Winter, Chris Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language Models are Few-Shot Learners. In *Advances in Neural Information Processing Systems (NeurIPS '20, Vol. 33)*. Curran Associates, Inc., Red Hook, NY, USA, 1877–1901.

[9] Tuhin Chakrabarty, Vishakh Padmakumar, and He He. 2022. Help me write a Poem - Instruction Tuning as a Vehicle for Collaborative Poetry Writing. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing (EMNLP '22)*. Association for Computational Linguistics, Online, 6848–6863. https://doi.org/10.18653/v1/2022.emnlp-main.460

[10] Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Pondé de Oliveira Pinto, Jared Kaplan, Harrison Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian, Clemens Winter, Philippe Tillet, Felipe Petroski Such, Dave Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William Hebgen Guss, Alex Nichol, Alex Paino, Nikolas Tezak, Jie Tang, Igor Babuschkin, Suchir Balaji, Shantanu Jain, William Saunders, Christopher Hesse, Andrew N. Carr, Jan Leike, Joshua Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. 2021. Evaluating Large Language Models Trained on Code. (2021). https://doi.org/10.48550/arxiv.2107.03374

[11] Hai Dang, Sven Goller, Florian Lehmann, and Daniel Buschek. 2023. Choice Over Control: How Users Write with Large Language Models using Diegetic and Non-Diegetic Prompting. In *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems (CHI '23)*. ACM, New York, NY, USA, 17 pages. https://doi.org/10.1145/3544548.3580969

[12] Sumanth Dathathri, Andrea Madotto, Janice Lan, Jane Hung, Eric Frank, Piero Molino, Jason Yosinski, and Rosanne Liu. 2020. Plug and Play Language Models: A Simple Approach to Controlled Text Generation. In *Proceedings of the 8th International Conference on Learning Representations (ICLR)*. OpenReview.net, Online.

[13] Graham Dove, Kim Halskov, Jodi Forlizzi, and John Zimmerman. 2017. UX Design Innovation: Challenges for Working with Machine Learning as a Design Material. In *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems (CHI '17)*. ACM, New York, NY, USA, 278–288. https://doi.org/10.1145/3025453.3025739

[14] Wanyu Du, Zae Myung Kim, Vipul Raheja, Dhruv Kumar, and Dongyeop Kang. 2022. Read, Revise, Repeat: A System Demonstration for Human-in-the-loop Iterative Text Revision. In *Proceedings of the 1st Workshop on Intelligent and Interactive Writing Assistants (In2Writing '22)*. Association for Computational Linguistics, Online, 96–108. https://doi.org/10.18653/v1/2022.in2writing-1.14

[15] Patrick M. J. Dubois, Volodymyr Dziubak, and Andrea Bunt. 2017. Tell Me More! Soliciting Reader Contributions to Software Tutorials. In *Proceedings of the 43rd Graphics Interface Conference (GI '17)*. Canadian Human-Computer Communications Society/ACM, New York, USA, 16–23. https://doi.org/10.20380/GI2017.03

[16] Linda Flower and John R. Hayes. 1981. A Cognitive Process Theory of Writing. *College Composition and Communication* 32, 4 (1981), 365–387. https://doi.org/10.2307/356600

[17] Katy Gero, Alex Calderwood, Charlotte Li, and Lydia Chilton. 2022. A Design Space for Writing Support Tools Using a Cognitive Process Model of Writing. In *Proceedings of the 1st Workshop on Intelligent and Interactive Writing Assistants (In2Writing '22)*. Association for Computational Linguistics, Online, 11–24. https://doi.org/10.18653/v1/2022.in2writing-1.2

[18] Katy Ilonka Gero, Vivian Liu, and Lydia Chilton. 2022. Sparks: Inspiration for Science Writing using Language Models. In *Proceedings of the 2022 ACM Designing Interactive Systems Conference (DIS '22)*. ACM, New York, NY, USA, 1002–1019. https://doi.org/10.1145/3532106.3533533

[19] Maliheh Ghajargar, Jeffrey Bardzell, and Love Lagerkvist. 2022. A Redhead Walks into a Bar: Experiences of Writing Fiction with Artificial Intelligence. In *Proceedings of the 25th International Academic Mindtrek Conference (Academic Mindtrek '22)*. ACM, New York, NY, USA, 230–241. https://doi.org/10.1145/3569219.3569418

[20] Shiry Ginosar, Luis Fernando De Pombo, Maneesh Agrawala, and Björn Hartmann. 2013. Authoring multi-stage code examples with editable code histories. In *Proceedings of the 26th Annual ACM Symposium on User Interface Software and Technology (UIST '13)*. ACM, New York, NY, USA, 485–494. https://doi.org/10.1145/2501988.2502053

[21] Steven M. Goodman, Erin Buehler, Patrick Clary, Andy Coenen, Aaron Donsbach, Tiffanie N. Horne, Michal Lahav, Robert MacDonald, Rain Breaw Michaels, Ajit Narayanan, Mahima Pushkarna, Joel Riley, Alex Santana, Lei Shi, Rachel Sweeney, Phil Weaver, Ann Yuan, and Meredith Ringel Morris. 2022. LaMPost: Design and Evaluation of an AI-assisted Email Writing Prototype for Adults with Dyslexia. In *Proceedings of the 24th International ACM SIGACCESS Conference on Computers and Accessibility (ASSETS '22)*. ACM, New York, NY, USA, 24:1–24:18. https://doi.org/10.1145/3517428.3544819

[22] Mitchell L. Gordon and Philip J. Guo. 2015. Codepourri: Creating visual coding tutorials using a volunteer crowd of learners. In *2015 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC '15)*. IEEE Computer Society, 13–21. https://doi.org/10.1109/vlhcc.2015.7357193

[23] Greg Guest, Arwen Bunce, and Laura Johnson. 2006. How Many Interviews Are Enough?: An Experiment with Data Saturation and Variability. *Field Methods* 18, 1 (2006), 59–82. https://doi.org/10.1177/1525822X05279903

[24] Andrew Head, Elena L. Glassman, Björn Hartmann, and Marti A. Hearst. 2018. Interactive Extraction of Examples from Existing Code. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems (CHI '18)*. ACM, New York, NY, USA, 1–12. https://doi.org/10.1145/3173574.3173659

[25] Andrew Head, Jason Jiang, James Smith, Marti A. Hearst, and Björn Hartmann. 2020. Composing Flexibly-Organized Step-by-Step Tutorials from Linked Source Code, Snippets, and Outputs. In *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems (CHI '20)*. ACM, New York, NY, USA, 1–12. https://doi.org/10.1145/3313831.3376798

[26] Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2022. LoRA: Low-Rank Adaptation of Large Language Models. In *Proceedings of the 10th International Conference on Learning Representations (ICLR)*. OpenReview.net, Online.

[27] Daphne Ippolito, Ann Yuan, Andy Coenen, and Sehmon Burnam. 2022. Creative Writing with an AI-Powered Writing Assistant: Perspectives from Professional Writers. (2022). https://doi.org/10.48550/arxiv.2211.05030

[28] Ziwei Ji, Nayeon Lee, Rita Frieske, Tiezheng Yu, Dan Su, Yan Xu, Etsuko Ishii, Ye Jin Bang, Andrea Madotto, and Pascale Fung. 2023. Survey of Hallucination in Natural Language Generation. *Comput. Surveys* 55, 12 (2023), 38 pages. https://doi.org/10.1145/3571730

[29] Amela Karahasanovic, Unni Nyhamar Hinkel, Dag I. K. Sjøberg, and Richard C. Thomas. 2009. Comparing of feedback-collection and think-aloud methods in program comprehension studies. *Behaviour & Information Technology* 28, 2 (2009), 139–164. https://doi.org/10.1080/01449290701682761

[30] Ada S. Kim and Amy J. Ko. 2017. A Pedagogical Analysis of Online Coding Tutorials. In *Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education (SIGCSE '17)*. ACM, New York, NY, USA, 321–326. https://doi.org/10.1145/3017680.3017728

[31] Max Kreminski and Chris Martens. 2022. Unmet Creativity Support Needs in Computationally Supported Creative Writing. In *Proceedings of the 1st Workshop on Intelligent and Interactive Writing Assistants (In2Writing '22)*. Association for Computational Linguistics, Online, 74–82. https://doi.org/10.18653/v1/2022.in2writing-1.11

[32] Benjamin Lafreniere, Tovi Grossman, and George Fitzmaurice. 2013. Community enhanced tutorials: improving tutorials with multiple demonstrations. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '13)*. ACM, New York, NY, USA, 1779–1788. https://doi.org/10.1145/2470654.2466235

[33] Mina Lee, Katy Ilonka Gero, John Joon Young Chung, Simon Buckingham Shum, Vipul Raheja, Hua Shen, Subhashini Venugopalan, Thiemo Wambsganss, David Zhou, Emad A. Alghamdi, Tal August, Avinash Bhat, Madiha Zahrah Choksi, Senjuti Dutta, Jin L. C. Guo, Md Naimul Hoque, Yewon Kim, Simon Knight, Seyed Parsa Neshaei, Antonette Shibani, Disha Shrivastava, Lila Shroff, Agnia Sergeyuk, Jessi Stark, Sarah Sterman, Sitong Wang, Antoine Bosselut, Daniel Buschek, Joseph Chee Chang, Sherol Chen, Max Kreminski, Joonsuk Park, Roy Pea, Eugenia Ha Rim Rho, Zejiang Shen, and Pao Siangliulue. 2024. A Design Space for Intelligent and Interactive Writing Assistants. In *Proceedings of the 2024 CHI Conference on Human Factors in Computing Systems (CHI '24)*. ACM, New York, NY, USA. https://doi.org/10.1145/3613904.3642697

[34] Mina Lee, Percy Liang, and Qian Yang. 2022. CoAuthor: Designing a Human-AI Collaborative Writing Dataset for Exploring Language Model Capabilities. In *Proceedings of the 2022 CHI Conference on Human Factors in Computing Systems (CHI '22)*. ACM, New York, NY, USA, 388:1–388:19. https://doi.org/10.1145/3491102.3502030

[35] Yoonjoo Lee, Tae Soo Kim, Minsuk Chang, and Juho Kim. 2022. Interactive Children's Story Rewriting Through Parent-Children Interaction. In *Proceedings of the 1st Workshop on Intelligent and Interactive Writing Assistants (In2Writing '22)*. Association for Computational Linguistics, Online, 62–71. https://doi.org/10.18653/v1/2022.in2writing-1.9

[36] Juho Leinonen, Paul Denny, Stephen MacNeil, Sami Sarsa, Seth Bernstein, Joanne Kim, Andrew Tran, and Arto Hellas. 2023. Comparing Code Explanations Created by Students and Large Language Models. In *Proceedings of the 2023 Conference on Innovation and Technology in Computer Science Education (ITiCSE 2023)*. ACM, New York, NY, USA, 124–130. https://doi.org/10.1145/3587102.3588785

[37] Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. 2020. Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks. In *Advances in Neural Information Processing Systems (NeurIPS '20, Vol. 33)*. Curran Associates, Inc., Red Hook, NY, USA, 9459–9474.

[38] Yang Li, Jiacong He, Xin Zhou, Yuan Zhang, and Jason Baldridge. 2020. Mapping Natural Language Instructions to Mobile UI Action Sequences. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*. Association for Computational Linguistics, Online, 8198–8210. https://doi.org/10.18653/v1/2020.acl-main.729

[39] Pengfei Liu, Weizhe Yuan, Jinlan Fu, Zhengbao Jiang, Hiroaki Hayashi, and Graham Neubig. 2023. Pre-train, Prompt, and Predict: A Systematic Survey of Prompting Methods in Natural Language Processing. *Comput. Surveys* 55, 9 (2023), 35 pages. https://doi.org/10.1145/3560815

[40] Michael Meng, Stephanie Steinhardt, and Andreas Schubert. 2018. Application Programming Interface Documentation: What Do Software Developers Want? *Journal of Technical Writing and Communication* 48, 3 (2018), 295–330. https://doi.org/10.1177/0047281617721853

[41] Michael Meng, Stephanie Steinhardt, and Andreas Schubert. 2019. How developers use API documentation: an observation study. *Communication Design Quarterly* 7, 2 (2019), 40–49. https://doi.org/10.1145/3358931.3358937

[42] Samantha B. Meyer and Belinda Lunnay. 2013. The Application of Abductive and Retroductive Inference for the Design and Analysis of Theory-Driven Sociological Research. *Sociological Research Online* 18, 1 (2013), 86–96. https://doi.org/10.5153/sro.2819

[43] Piotr Mirowski, Kory W. Mathewson, Jaylen Pittman, and Richard Evans. 2023. Co-Writing Screenplays and Theatre Scripts with Language Models: Evaluation by Industry Professionals. In *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems (CHI '23)*. ACM, New York, NY, USA, 355:1–355:34. https://doi.org/10.1145/3544548.3581225

[44] Margaret Mitchell, Simone Wu, Andrew Zaldivar, Parker Barnes, Lucy Vasserman, Ben Hutchinson, Elena Spitzer, Inioluwa Deborah Raji, and Timnit Gebru. 2019. Model Cards for Model Reporting. In *Proceedings of the Conference on Fairness, Accountability, and Transparency (FAT* '19)*. ACM, New York, NY, USA, 220–229. https://doi.org/10.1145/3287560.3287596

[45] Alok Mysore and Philip J. Guo. 2017. Torta: Generating Mixed-Media GUI and Command-Line App Tutorials Using Operating-System-Wide Activity Tracing. In *Proceedings of the 30th Annual ACM Symposium on User Interface Software and Technology (UIST '17)*. ACM, New York, NY, USA, 703–714. https://doi.org/10.1145/3126594.3126628

[46] Jakob Nielsen and Thomas K. Landauer. 1993. A mathematical model of the finding of usability problems. In *Proceedings of the INTERACT '93 and CHI '93 Conference on Human Factors in Computing Systems (INTERCHI)*. ACM, 206–213. https://doi.org/10.1145/169059.169166

[47] OpenAI. 2021. OpenAI Codex, openai.com. https://openai.com/blog/openai-codex. [Accessed 26-04-2024].

[48] OpenAI. 2022. OpenAI Playground, openai.com. https://platform.openai.com/playground/complete. [Accessed 03-05-2024].

[49] OpenAI. 2023. OpenAI Deprecations, openai.com. https://platform.openai.com/docs/deprecations. [Accessed 03-05-2024].

[50] Chris Parnin, Christoph Treude, and Margaret-Anne D. Storey. 2013. Blogging developer knowledge: Motivations, challenges, and future directions. In *Proceedings of the IEEE 21st International Conference on Program Comprehension (ICPC)*. IEEE Computer Society, 211–214. https://doi.org/10.1109/icpc.2013.6613850

[51] Dewayne E. Perry, Nancy A. Staudenmayer, and Lawrence G. Votta. 1994. People, Organizations, and Process Improvement. *IEEE Software* 11, 4 (1994), 36–45. https://doi.org/10.1109/52.300082

[52] Martin P. Robillard, Deeksha M. Arya, Neil A. Ernst, Jin L.C. Guo, Maxime Lamothe, Mathieu Nassif, Nicole Novielli, Alexander Serebrenik, Igor Steinmacher, and Klaas-Jan Stol. 2024. Communicating Study Design Trade-offs in Software Engineering. *ACM Transactions on Software Engineering and Methodology* (2024). https://doi.org/10.1145/3649598

[53] Melissa Roemmele. 2021. Inspiration through Observation: Demonstrating the Influence of Automatically Generated Text on Creative Writing. In *Proceedings of the 12th International Conference on Computational Creativity (ICCC)*. Association for Computational Creativity (ACC), 52–61.

[54] Steven I. Ross, Fernando Martinez, Stephanie Houde, Michael J. Muller, and Justin D. Weisz. 2023. The Programmer's Assistant: Conversational Interaction with a Large Language Model for Software Development. In *Proceedings of the 28th International Conference on Intelligent User Interfaces (IUI '23)*. ACM, New York, NY, USA, 491–514. https://doi.org/10.1145/3581641.3584037

[55] Interagency Language Roundtable. 2021. Interagency Language Roundtable Language Skill Level Descriptions, govtilr.org. https://www.govtilr.org/Skills/ILRscale2.htm. [Accessed 19-05-2024].

[56] Carolyn B. Seaman. 2008. Qualitative Methods. In *Guide to Advanced Empirical Software Engineering*. Springer, 35–62. https://doi.org/10.1007/978-1-84800-044-5_2

[57] Hua Shen, Chieh-Yang Huang, Tongshuang Wu, and Ting-Hao Kenneth Huang. 2023. ConvXAI: Delivering Heterogeneous AI Explanations via Conversations to Support Human-AI Scientific Writing. In *Companion Publication of the 2023 Conference on Computer Supported Cooperative Work and Social Computing (CSCW '23 Companion)*. ACM, New York, NY, USA, 384–387. https://doi.org/10.1145/3584931.3607492

[58] Zejiang Shen, Tal August, Pao Siangliulue, Kyle Lo, Jonathan Bragg, Jeff Hammerbacher, Doug Downey, Joseph Chee Chang, and David A. Sontag. 2023. Beyond Summarization: Designing AI Support for Real-World Expository Writing Tasks. (2023). https://doi.org/10.48550/arxiv.2304.02623

[59] Disha Shrivastava, Hugo Larochelle, and Daniel Tarlow. 2023. Repository-Level Prompt Generation for Large Language Models of Code. In *Proceedings of the 40th International Conference on Machine Learning (Proceedings of Machine Learning Research, Vol. 202)*. PMLR, 31693–31715. https://proceedings.mlr.press/v202/shrivastava23a.html

[60] Janice Singer, Susan Elliott Sim, and Timothy C. Lethbridge. 2008. Software Engineering Data Collection for Field Studies. In *Guide to Advanced Empirical Software Engineering*. Springer, 9–34. https://doi.org/10.1007/978-1-84800-044-5_1

[61] Karen M Staller. 2021. Big enough? Sampling in qualitative inquiry. *Qualitative Social Work* 20, 4 (2021), 897–904. https://doi.org/10.1177/14733250211024516

[62] Hariharan Subramonyam, Christopher Lawrence Pondoc, Colleen M. Seifert, Maneesh Agrawala, and Roy Pea. 2023. Bridging the Gulf of Envisioning: Cognitive Design Challenges in LLM Interfaces. (2023). https://doi.org/10.48550/arxiv.2309.14459

[63] Jon Swain. 2018. *A Hybrid Approach to Thematic Analysis in Qualitative Research: Using a Practical Example*. SAGE Publications Ltd. https://doi.org/10.4135/9781526435477

[64] Lev Tankelevitch, Viktor Kewenig, Auste Simkute, Ava Elizabeth Scott, Advait Sarkar, Abigail Sellen, and Sean Rintel. 2023. The Metacognitive Demands and Opportunities of Generative AI. (2023). https://doi.org/10.48550/arxiv.2312.10893

[65] Priyan Vaithilingam, Tianyi Zhang, and Elena L. Glassman. 2022. Expectation vs. Experience: Evaluating the Usability of Code Generation Tools Powered by Large Language Models. In *Extended Abstracts of the 2022 CHI Conference on Human Factors in Computing Systems (CHI '22)*. ACM, New York, NY, USA, 332:1–332:7. https://doi.org/10.1145/3491101.3519665

[66] April Yi Wang, Andrew Head, Ashley Ge Zhang, Steve Oney, and Christopher Brooks. 2023. Colaroid: A Literate Programming Approach for Authoring Explorable Multi-Stage Tutorials. In *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems (CHI '23)*. ACM, New York, NY, USA, 798:1–798:22. https://doi.org/10.1145/3544548.3581525

[67] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed Chi, Quoc V Le, and Denny Zhou. 2022. Chain-of-Thought Prompting Elicits Reasoning in Large Language Models. 35 (2022), 24824–24837.

[68] Daijin Yang, Yanpeng Zhou, Zhiyuan Zhang, Toby Jia-Jun Li, and Ray LC. 2022. AI as an Active Writer: Interaction Strategies with Generated Text in Human-AI Collaborative Fiction Writing. In *Joint Proceedings of the IUI 2022 Workshops: APEx-UI, HAI-GEN, HEALTHI, HUMANIZE, TExSS, SOCIALIZE co-located with the ACM International Conference on Intelligent User Interfaces (IUI) (CEUR Workshop Proceedings, Vol. 3124)*. CEUR-WS.org, Online, 56–65.

[69] Qian Yang, Aaron Steinfeld, Carolyn P. Rosé, and John Zimmerman. 2020. Re-examining Whether, Why, and How Human-AI Interaction Is Uniquely Difficult to Design. In *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems (CHI '20)*. ACM, New York, NY, USA, 1–13. https://doi.org/10.1145/3313831.3376301

[70] Nur Yildirim, Changhoon Oh, Deniz Sayar, Kayla Brand, Supritha Challa, Violet Turri, Nina Crosby Walton, Anna Elise Wong, Jodi Forlizzi, James McCann, and John Zimmerman. 2023. Creating Design Resources to Scaffold the Ideation of AI Concepts. In *Proceedings of the 2023 ACM Designing Interactive Systems Conference (DIS '23)*. ACM, New York, NY, USA, 2326–2346. https://doi.org/10.1145/3563657.3596058

[71] 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)*. ACM, New York, NY, USA, 437:1–437:21. https://doi.org/10.1145/3544548.3581388

[72] Mingyuan Zhong, Gang Li, Peggy Chi, and Yang Li. 2021. HelpViz: Automatic Generation of Contextual Visual Mobile Tutorials from Text-Based Instructions. In *The 34th Annual ACM Symposium on User Interface Software and Technology (UIST '21)*. ACM, New York, NY, USA, 1144–1153. https://doi.org/10.1145/3472749.3474812