Investigating Explainability of Generative AI for Code through Scenario-based Design

Jiao Sun*
University of Southern California
Los Angeles, USA
jiaosun@usc.edu

Q. Vera Liao†
Microsoft Research
Montréal, Canada
veraliao@microsoft.com

Michael Muller
IBM Research AI
Yorktown Heights, USA
michael_muller@us.ibm.com

Mayank Agarwal
IBM Research AI
Yorktown Heights, USA
Mayank.Agarwal@ibm.com

Stephanie Houde
IBM Research AI
Yorktown Heights, USA
Stephanie.Houde@ibm.com

Kartik Talamadupula
IBM Research AI
Yorktown Heights, USA
krtalamad@us.ibm.com

Justin D. Weisz
IBM Research AI
Yorktown Heights, USA
jweisz@us.ibm.com

ABSTRACT

What does it mean for a generative AI model to be *explainable*? The emergent discipline of explainable AI (XAI) has made great strides in helping people understand discriminative models. Less attention has been paid to generative models that produce *artifacts*, rather than *decisions*, as output. Meanwhile, generative AI (GenAI) technologies are maturing and being applied to application domains such as software engineering. Using scenario-based design and question-driven XAI design approaches, we explore users' explainability needs for GenAI in three software engineering use cases: natural language to code, code translation, and code auto-completion. We conducted 9 workshops with 43 software engineers in which real examples from state-of-the-art generative AI models were used to elicit users' explainability needs. Drawing from prior work, we also propose 4 types of XAI features for GenAI for code and gathered additional design ideas from participants. Our work explores explainability needs for GenAI for code and demonstrates how human-centered approaches can drive the technical development of XAI in novel domains.

CCS CONCEPTS

• **Computing methodologies** → **Natural language generation**; • **Software and its engineering**; • **Human-centered computing** → *Human computer interaction (HCI)*; *User studies*;

---

*Work done during the first author's internship at IBM Research AI.
†Work done while the second author was at IBM Research AI.

KEYWORDS

generative AI, software engineering tooling, explainable AI, human-centered AI, scenario based design

1 INTRODUCTION

Generative AI (GenAI) is a class of machine learning (ML) algorithms that can learn from content such as text, images, and audio in order to generate new content. In contrast to discriminative ML algorithms, which learn decision boundaries, GenAI models produce artifacts as output, which can have a wide range of variety and complexity. One major recent development of GenAI is the introduction of OpenAI's GPT-3 [13] model, which can generate human-like language output and has striking versatility. Other generative language models have emerged that focus on specific domains such as software engineering, implementing use cases of auto-completing code [15, 39], translating code from one programming language to another [88], and converting natural language to code [23]. The industry has begun to use these models to support software engineering practices, with the most prominent example being GitHub CoPilot [27], a GenAI-based co-programming tool.

As a novel technology applied to novel domains, there are many open questions to be answered for how to make GenAI more capable and user-friendly. One open question is how to enable *explainability*—allowing users to understand and have a better mental model—of GenAI. Recent works by Goodfellow et al. [28] and Ross et al. [86] have explored developing more interpretable GenAI models that follow more human-understandable processes. However, a more comprehensive view of explainability for GenAI is lacking: what do users need to understand about a GenAI model in order to

effectively achieve their goals when working with it? In this paper, we build a foundational understanding of explainability needs for GenAI in the context of generative code models.

This question of *what do users need to understand about AI systems* is core to the nascent field of Human-Centered Explainable AI (HCXAI) [21, 22, 52], which is a subset of the fields of human centered AI and human centered data science [6, 7, 25, 41, 65–67]. Our work is informed by a few key lessons from recent work in HCXAI, mostly conducted in the context of discriminative ML (e.g., for decision-support systems). First, explainability needs should be considered broadly as *any means of helping users achieve a better understanding of the AI system.* Liao et al. [48] proposed to define user's explainablity needs by what questions they ask to understand the AI [48] and developed a framework of common questions. This framework demonstrates that users are interested in a broad range of explanatory information about an AI model, including its overall logic, how it reasons to produce a particular output, the training data, and its performance and range of output. However, user needs regarding *generative* models were not explored in that work.

Second, XAI solutions that address explainability needs *should not be limited to algorithmic explanations or showing model internals.* Depending on user needs, it may be more critical to provide transparent information about a model's capabilities, limitations (e.g. uncertainty [10]) or provenance [8]. Moreover, users may need additional information beyond algorithmic explanations to fill in gaps of understanding. For example, Ehsan et al. [19] proposed that *social transparency* – making visible the socio-organizational factors that govern the use of AI – can help users form a socially-situated understanding of an AI system and take more effective actions with it.

Finally, perhaps the most important lesson from HCXAI is that users' explainability needs *emerge in a usage context*, guided by their goals and shaped by their backgrounds, expectations, as well as socio, organizational and cultural contexts [20, 48, 52]. It is thus necessary to follow a user-centered approach to understand explainability needs by involving target users and leveraging HCI methods that allow inquiry within the context of usage.

Based on these lessons, we adopted a scenario-based design method by constructing realistic usage scenarios for three use cases of GenAI for code: code translation, code auto-completion, and natural language to code. We invited 43 software engineers to participate in 9 workshops to elicit their explainability needs and design ideas around these scenarios. We adapted the question-driven method of Liao et al. [48, 50] to elicit and comprehensively explore participants' explainability needs by what kind questions they would ask in the scenarios. We also gathered feedback and design ideas from participants for four kinds of XAI features that we propose for the uses cases of GenAI for code. Our work makes three main contributions to the IUI community:

(1) We identify 11 categories of explainability needs in the context of Generative AI (GenAI) for code, for which we provide definitions and examples. We further contrast these categories with previous XAI techniques for discriminative ML and discuss explainability needs unique to GenAI and code generation use cases. We believe we are among the first to explore users' explainability needs in an application domain of GenAI.

(2) We propose four kinds of XAI features to support users of GenAI for code, based on prior work and adapted to the domain of code generation. These features are: AI documentation, indications of model uncertainty, visualizations of model attention, and social transparency. Based on participants' responses, we provide concrete design recommendations to operationalize these features.

(3) Our work makes methodological contributions by combining scenario-based design, participatory design workshops, and a question-driven approach to elicit explainability needs. We also reflect on the values and limitations of this method to inform future work that explores GenAI in new domains.

2 RELATED WORK

We review three areas of related work that shaped our study: GenAI for code, explainable AI, and human-centered approaches to AI.

2.1 Generative AI for Code

The application of modern NLP techniques to programming language can be traced back to the *naturalness hypothesis* [4, 17, 37], that software is a form of human communication. This hypothesis opened the door for applying NLP techniques previously used on human natural languages to source code, and recent work in this space is summarized by Talamadupula [92] and Allamanis et al. [4]. One example is how existing work on automatic, machine translation between human natural languages [71, 72] was applied to code. Specifically, the TransCoder [88] system applied neural machine translation techniques to translate source code across different languages. Other GenAI models have been developed that implement other use cases, such as generating documentation for code [23], auto-completing code [15, 39], generating unit tests [93], and finding duplicate code [30]. Models trained on massive code data sets are even able to handle multiple use cases at the same time, such as PLBART [3], CodeBERT [23], GraphCodeBERT [30]. Most recently, OpenAI released Codex [15], which is a GPT-based model trained on code from GitHub and powers their CoPilot [27] product. This model is capable of auto-completing code for various programming languages (e.g., Python, TypeScript, Go, Ruby), as well as generating code from natural language. The release of Copilot is seen as a revolution of AI-assisted software programming and has attracted much attention since its release [63].

Despite the fact that GenAI for code models still have room for improvement in the quality of their outputs – e.g., TransCoder only produces a correct translation 30%-70% of the time depending on the source and target language [88] – recent work by Weisz et al. [100] suggests that software engineers may nonetheless be tolerant of using such models in their work. Given the emerging productization of GenAI for code models, we believe it is necessary to develop a comprehensive understanding of the kinds of questions software engineers will have when working with such models to guide technical XAI work and design solutions to answer them.

2.2 Explainable AI

Explainable AI (XAI) has spurred great academic, industry, and public interests in the past few years, thanks to the popularity of inscrutable "opaque-box" ML models. Many XAI techniques have

been developed for discriminative ML models, both through producing directly interpretable models [14, 44] and generating post-hoc explanations for a trained opaque-box model [47, 62, 80, 104]. These explanations take a variety of forms. For example, *global* explanations provide an overview of the model logic, while *local* explanations elucidate the rationale behind a particular output. A full review of XAI techniques is beyond the scope of this paper and can be found in many recent survey papers [1, 29, 55, 56]. Our work is most closely informed by, and intends to bridge, the emerging topic of explainability for generative models, and the inter-disciplinary field of Human-Centered Explainable AI (HCXAI) [21, 22, 52].

Compared to discriminative models, much less attention has been paid to developing XAI techniques for generative AI models. Some explored ways to make generative models more directly interpretable, which is often framed as making the representation learning of a GenAI model in its latent dimensions semantically meaningful so people can directly examine the model internals [28]. For example, disentanglement is a technique that seeks mappings between high-dimensional inputs and low-dimensional representations such that representation dimensions correspond to the ground-truth factors that generated the data [83]. Accordingly, some proposed disentanglement measures [16, 84] as a way to evaluate a GenAI model's interpretability. A recent study by Ross et al. [86] proposed a user evaluation task to evaluate the human interpretability of generative models, by the ability for people to interactively modify representations to reconstruct target instances.

Others explored visualization approaches to present the representations to help users (often model developers) make sense of what the GenAI model has learned. For example, Ross et al. [86] use sliders to let users dynamically modify representation dimensions and see how corresponding instances change. Recent HCI works also explored the approach of "explainability through interaction" for generative models, by allowing users to interact with the input or guiding output generation process [59, 60, 86, 103]. Through observing immediate feedback from changes to the model output, people can make better sense of how a generative model works. For example, Zhang and Banovic developed a system that allows users to interactively explore the output space of an image generative model to assess the model quality [103].

These varied approaches suggest that explainability is still a less than well-defined notion for GenAI. Here we adopt a HCXAI position that explainability, or the effectiveness of explanation, should be defined as enabling people's understanding of the AI to achieve their goals [52]. With this human-centered definition, many have argued that it is necessary to provide transparent information beyond the model internals, such as its performance, limitations, training data, and development procedure, to enable a holistic understanding of the AI and more actionable insights [11, 48, 73, 94]. As discussed, Liao et al. proposed to identify users' explainability needs by eliciting the questions they ask to understand the AI [48, 50]. This method allows approaching XAI from the users' perspective and thoroughly identify transparent information they need to achieve an understanding necessary for their goals, which, in the context of GenAI for code, could be about optimizing for the usage of the AI system and overall productivity. Liao et al.'s work was based on prior HCI work that defines "intelligibility types" of information for context-aware intelligent systems by prototypical questions such

as Input, Output, Why, What-if, etc. [53, 54]. It also draws on social science literature showing that people's explanatory goals can be expressed in different kinds of questions [34]. We build on this line of work and adapt the question-drive method to elicit users' explainability needs for GenAI for code.

2.3 Human-centered approaches to AI

The term Human-centered AI has emerged in many academic works and public discussions [22, 25, 45, 65, 85, 89]. While definitions vary, human-centered approaches to AI aim to develop AI systems that serve the needs, improve the conditions, and align with the values of human stakeholders. Research begins to develop practical methods that can help achieve these goals. One set of methods gained much attention lately under the umbrella term of "participatory machine learning" [32, 46, 97]. Built on the tradition of participatory design, participatory machine learning emphasizes involving stakeholders, especially affected marginalized groups, into the development process early on to shape the overall goals and design choices of ML systems. Some suggested that abstracting user values from their participatory input is especially effective to guide ML modeling choices such as defining its optimization functions, features, constraints, and so on [49, 68, 105]. This notion of driving technical development based on insights from empirical user studies is also at the core of broad IUI research [5].

One challenge to involving users to shape the design and technical development of AI systems is that these systems often do not exist yet for people to experience and provide realistic feedback. This challenge can be tackled by human-centered methods that allow "envisioning future use possibilities". One effective method is scenario-based design (SBD) [87]. SBD suspends the needs to define system operations by using narrative descriptions of how a user uses a system to accomplish a task, allowing people to respond to concrete interactions. We chose to use SBD to explore GenAI for code use cases as most software engineers do not have experience with such technologies. SBD also allows us to explore XAI design without the constraint of current technical feasibility, as adopted by several prior XAI works [19, 102].

3 METHODOLOGY: SCENARIO-BASED DESIGN WORKSHOPS

We conducted 9 semi-structured workshops, with 3-6 participants in each, which lasted for 60-70 minutes. Due to the impact of the global COVID-19 pandemic, participants joined the workshop remotely via a video conferencing tool. We also used Mural[1], which provides visual workspaces for virtual collaboration. Each workshop was based on one of three use cases of GenAI for code: code translation, code autocompletion, and natural language to code. In the following, we first introduce the three use cases then describe the workshop in detail, then participants and analysis.

3.1 Use Cases and Scenarios

We focus on three specific use cases of GenAI for code, based on the preferred choices that could deliver high value for software

---

[1]https://www.mural.co/

engineering tasks from a pre-study survey with 81 people who responded to our study recruitment message.

- *Code translation*, in which a generative model translates source code from one language (e.g., Java) to another (e.g., Python). This task has been an important benchmark for technical work in GenAI for code [61] and has gained extensive attention from both industry and academia [3, 23, 30, 88, 99]. Such technologies can significantly reduce the cost and expertise barriers for code modernization work, in which a legacy codebase is ported to a modern programming language.
- *Code autocompletion*, in which a generative model takes comments and source code as input (e.g. a function specification and/or signature), and produces code as output (e.g. the implementation of the function). This use case can fulfill pervasive needs of software engineers to improve their productivity and efficiency. Notably, autocompletion is one of the primary functions of GitHub Copilot.
- *Natural language to code*, in which a generative model takes natural language (e.g. "change the color of the button to blue") and produces code as output (e.g. `button.setColor(Color.blue)`). This use case is another function offered by GitHub Copilot and represents a promising way to reduce entry barriers to programming.

For each use case, we created a persona, Alex, who is a software engineer working in a team. In the scenario given to participants, Alex and his teammates are introduced to an AI system to support their work. They are told that they could ask for more information and new functions added to the AI system to help them understand and work better with the AI. Figure 1 (*b*) shows the description of the persona for the code auto-completion use case. A concrete scenario of a programming task that Alex would perform is shown in Figure 1 (*c*), and we show the programming tasks of the other two use cases (i.e., code translation and code autocompletion) in Appendix A.

To give participants a realistic experience, all AI-produced code in our scenarios was generated using state-of-the-art generative code models. We used TransCoder [88] for the code translation use case and Copilot [27] for the other two use cases. For the code translation use case, we selected a programming solution to the problem of converting integers to their Roman numeral representations.[2] One reason we selected this code example is that there was a subtle bug in the translation generated by TransCoder [88]. This bug was pointed out by the workshop facilitator when introducing the scenario, to allow us to probe on participants' reactions to the limitations of the AI. For the code autocompletion and natural language to code use cases, we selected a programming solution to the problem of counting the number of prime numbers from a given input. We sampled this problem from Project CodeNet [77], a large dataset of code samples. Based on CodeNet's metadata, the acceptance rate of Python solutions for this problem is lower than 50%, indicating that it is a non-trivial coding problem for which GenAI models might provide assistance.

---

[2]We selected a Java implementation of this problem from https://algorithms.tutorialhorizon.com.

3.2 Workshop Format and Procedure

In each workshop, the facilitator first stated the purpose of the workshop, introduced concepts about generative AI for code, and asked for participants' verbal consent and permission to record. Then, the facilitator started the recording and introduced the Alex persona and one of the programming problem scenarios described above. The main part of the workshop was carried out in two stages: first, an open-ended question elicitation exercise to understand participants' explainability needs in the given scenario; and second, an ideation session in which 4 types of XAI features were used as design probes to elicit feedback and design ideas, to be described below. The content of the persona, scenario, and design probes were customized based on which use case was selected for the workshop. All discussions and screen activities were recorded, transcribed, and analyzed, together with the content on the Mural pages.

3.2.1 Stage 1: Question elicitation exercise. The first stage of the workshop was designed to elicit what kinds of questions participants would have for understanding the GenAI in the given scenario, based on the question-driven approach to XAI design proposed by Liao et al. [48, 50]. After describing Alex's persona and the task scenario, the workshop facilitator asked participants: *"Put yourself in Alex's shoes, what does Alex want to know? What questions does Alex want to ask about the natural language to code model/AI?"* Participants were given several minutes to come up as many questions they could, posted as sticky notes on the Mural page. Next, the facilitator worked with participants to collaboratively cluster similar questions together. Participants were asked to speak out loud as they moved around the sticky notes to describe their thought process and rationale. This process encouraged participants to read each other's questions, ask for clarifications, and have discussions. After the clustering, the facilitator chose one sticky note in each cluster to represent the cluster, then invited participants to cast up to 3 votes for questions they considered to be the most important to address. After the voting session, participants were asked to share their rationales behind the votes to exchange ideas and stimulate new ideas.

3.2.2 Stage 2: Ideation on XAI features. In addition to exploring general explainability needs, our study also aims to explore a few areas to develop XAI features that can address these needs. Given the novelty in both the technical XAI approaches for GenAI and the application domain, we adapted existing ideas from XAI solutions for discriminative ML models as a starting point of our discussions. Liao et al. [50] provides a suggested mapping between prototypical user questions (e.g., why, performance, data, output) and XAI methods or features that can answer the questions, primarily in the context of discriminative AI for decision-support systems. We built upon this work, as well as other work that explores XAI or transparency features [10, 21, 31], to select and adapt features that could apply to the context of GenAI for code. We also took into consideration their technical feasibility and potential values they can provide for the use cases. Through this process, we arrived at the following XAI features:

- *AI documentation* is embodied by the recently-proposed concepts of FactSheets [8, 81], Model Cards [98], and About ML pages [78]. AI documentation is developed by AI model or
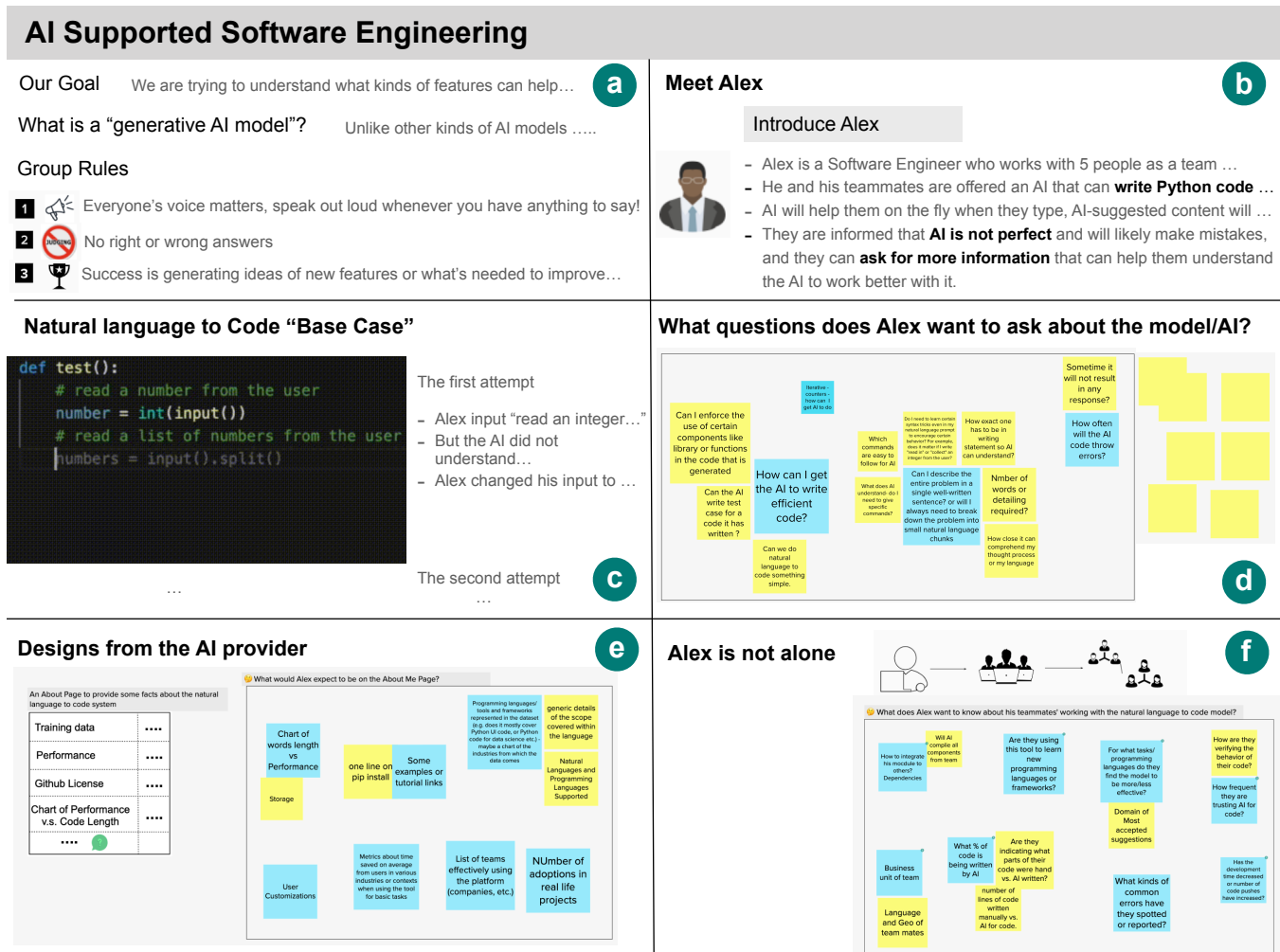
Figure 1: Overview of the Mural used in workshop #6 (W6-NL2Code) for the code autocompletion use case. We introduce the workshop and set ground rules in (a), introduce the Alex persona in (b), and show the model output for the use case in (c). Then, we ask participants for what they want to know about the GenAI model in (d), elicit design ideas for AI documentation in (e), and ask participants to ideate on what information Alex would want to know about his team members' use of the model in (f).

system providers to give users information about its purpose, performance, safety, and provenance. Liao et al. [48] suggest that this type of feature could be used to address several categories of explainability needs, including data, output, performance, and how (global logic) questions. We designed a table with 4 categories of facts about the model, without providing specific information, as shown in Figure 2 (a). This table served as a probe to elicit other categories of information that should be included in AI documentation for generative code models. We intentionally left the content incomplete to inspire design ideation from participants.

- *Uncertainty indicator* is a feature to communicate the model's uncertainty level about its output. Uncertainty is considered a critical form of transparency that can alert people about the limitations of AI [10]. In discriminative ML, uncertainty

can take the form of a confidence score for a classification model or a prediction interval for a regression model [26]. In the context of generative code models, we envisioned that uncertainty could be assessed and represented at the line level. This observation was motivated by a recent work by Agarwal et al. [2], which demonstrates how to derive line-level confidences by aggregating token probabilities. In the design probe (Figure 2 (b)), we communicate the line-level lack-of-confidence with an wavy underline when it falls below some uncertainty threshold.

- *Attention visualization* is a type of local explanation to answer the *why* question [51]: why did the model produce a particular output for a given input? For NLP tasks using deep neural networks, a common approach to explain a prediction is to utilize the attention mechanism, which provides

(a) AI documentation

(b) Uncertainty Indicator

(c) Attention Visualizer

(d) Social Transparency
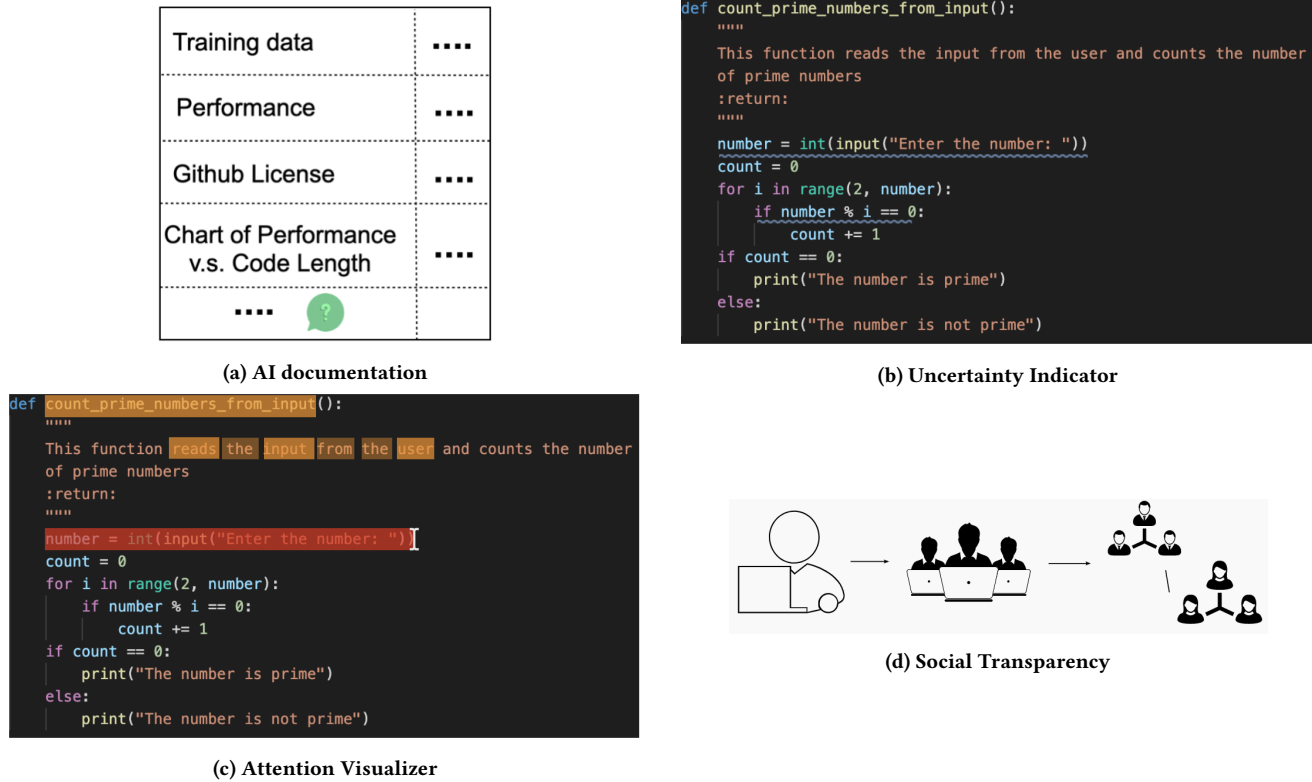
Figure 2: Examples of UI probes used in the code translation use case. Probes used in the other use cases were similar in design but differed in the specific code examples used. Each design is translated from existing XAI approaches for discriminative AI. The probes were designed to elicit questions and spark discussion around Alex's information needs for working with a generative code model.

a distribution over attended-to input units [101] to indicate the relative importance of input units for the output. Prior work has shown how to use attention weights to visually depict which words in an input sequence, plus which words in previously-generated output, were responsible for a given token's output [95, 96]. In our design probe (Figure 2 (c)), we show the user querying a portion of output (in red) and seeing which tokens were responsible for that output (in yellow, where opacity is used to signal the strength of the attention weight). The opacities in the figure are based on the actual cross-attention weights from the model we used to generate the output.

- *Social transparency* makes transparent the social contexts around the usage of an AI system to help people better understand the AI and how to utilize it. This feature was inspired by a recent work by Ehsan and Riedl [21], which examined the impact of making other users' past interactions with a decision-support AI system transparent. To explore social transparency in our context, we created an open-ended probe to invite ideation by visually emphasizing the fact that Alex is not working alone, but works within a group of other software engineers (Figure 2 (d)).

3.3 Participants

Recruitment. We advertised our study within a large international information technology company, by distributing a survey targeted at software engineers working across many product lines and locations. 81 people responded. In the recruiting survey, we asked questions about their demographic information, self evaluation of programming skills and previous experience of working with AI. In addition, we asked them to rate their interest in the 6 candidate user cases of generative AI models for code (in addition to the 3 selected use cases we included test case generation, documentaion generation and code repair).

Selection Criteria. For *code autocompletion* and *natural language to code* use cases, we required participants to have 1+ year of Python programming experience. For the *code translation* use case, we required participants to have 1+ year of both Python and Java programming experience. Among all sign-up responses, 56 (69%) had at least 1+ year of Python programming experience. From this set, we prioritized to select those who have had experience with AI systems, and we further narrowed down to 43 participants based on their time availability.

Demographics. For the geographical location, the majority of our participants were from the United States (N=22; 65%) and India

ID	Use Case	Participants
1	Code Translation (CT)	Software Engineer (P16, P20), Researcher/Scientist (P5, P52)
2	Code Translation (CT)	Software Engineer (P3, P62), Hardware Verification Engineer (P54), Data Scientist (P54), Researcher/Scientist (P25)
3	Code Autocompletion (CA)	Software Engineer (P2, P53 , P63), Data Engineer (P64), Researcher/Scientist (P66)
4	Code Autocompletion (CA)	Software Engineer (P17, P39), Researcher/Scientist (P37, P55), Bioinformatician (P48), Data Scientist (P61)
5	Code Translation (CT)	Researcher/Scientist (P19), Software Engineer (P46), Software Architect (P14, P45, P75), Data Scientist (P78)
6	Natural language to code (NL2Code)	Software Engineer (P44), Researcher/Scientist (P71), Data Scientist (P80, P81)
7	Natural language to code (NL2Code)	Software Engineer (P59, P15), Software Architect (P41, P42), Researcher/Scientist (P13), Data Scientist (P30)
8	Natural language to code (NL2Code)	Software Engineer (P7, P22), Data Scientist (P8, P32)
9	Code Autocompletion (CA)	Software Engineer (P57, P72), Data Engineer (P33)

Table 1: Participants and their roles in our 9 brainstorming workshops about co-designing generative AI models for Code. Cyan indicates the participant does not have experience of working with AI. We will use abbreviations to refer to workshops throughout the paper, for example, W1-CT for workshop 1 about the CT (Code Translation) use case.

(N=14; 41%), with smaller numbers from Canada (N=3; 7%), Germany (N=2; 5%), Ireland (N=1; 2%) and Brazil (N=1; 2%). For the gender identity, 33 (77%) participants identified as male, 8 (19%) female, and 2 (5%) preferred not to answer. Our participants also had diverse job roles: 17 (40%) software engineers, 9 (21%) researchers/scientists, 9 (21%) data scientists, 5 (12%) software architects, 1 data engineer, 1 bioinformatician and 1 hardware verification engineer. Among them, 40 (93%) had experience working with AI in their jobs, while 3 of them did not.

Assignment. We then assigned participants to 9 workshops (3 for each use case) as shown in Table 1, where we mark participants who do not have experience working with AI in blue.

3.4 Analysis

We conducted content analysis on the written content in Mural, and around 600 minutes of interviews were recorded and transcribed. In total, we extracted 487 messages from Mural and 249 paragraphs from the video transcript relevant to explainability. We followed the workshop structure, and coded the data in five parts: question elicitation exercise, discussions around AI documentation, uncertainty indicator, attention visualiser and social transparency.

To analyze the questions collected from the question elicitation exercise, we referred to prior work using prototypical questions to represent explainability needs by Liao et al. [48] in the context of discriminative ML systems, and "intelligibility types" [53, 54] for context-aware intelligent systems by Lim and Dey, and used these existing definitions to guide our coding. We also paid attention to new types of questions that were not covered in these prior works. Two researchers first independently coded 106 questions collected, and discussed their codes to reach a consensus. Then, they updated

their coding independently with the agreed codes. As a result, they reached a high level of agreement on 66 out of 71 messages (93%). After another iteration of discussion to finalize the coding schema, one of the two researchers coded the rest of questions. We discuss findings from this analysis in Section 4.

For the rest of the workshops, we coded participants' feedback and design ideas with regard to the four design probes. Our goal is to further understand users' explainability needs in the context of generative AI, as well as to inform concrete XAI designs that can address these needs. This part of coding was conducted by one researcher with frequent discussions with the other researchers. We discuss these findings in Section 5.

4 EXPLANABILTY NEEDS FOR GENAI FOR CODE

This section presents the results from the question elicitation exercise to understand what types of explainability needs particpants had for the GenAI for code use cases. As mentioned, we followed the definitions of prototypical user questions in XAI Question Bank by Liao et al. [48] and "intelligibility types" in Lim and Dey [53] to code the following categories: Input, Output, Performance, Data, How (global), Why/Why NOT, How to, What if, Control. We further identified two new categories that were not covered by the prior works: System Requirements and Impact, and Limitations. In Table 2, we present these question categories, with their definitions, and example questions asked by participants. They are ordered by the frequency of being asked by participants. Categories unique to the GenAI for code are marked with a dagger (†) sign. Categories that appeared in Lim and Dey [53] but not Liao et al. [48] are marked with an asterisk (∗) sign. Below we enumerate each

Category & Description	Exemplar Questions
Input*. Questions about the kinds of input that the model can take or should be given.	- What kind of input can the AI reasonably generate code from? - What types of [data types, data structures, algorithm...] does the AI understand? - How specific does the input need to be to get a good output? - Is there certain thing I need to specify to get a good output?
Output. Questions about what the AI can produce, such as the output type, scope and system capabilities.	- Will this produce idiomatic Python code, or a 1:1 translation? - Will the translated code have the same [variable name, complexity, etc.] as the original? - Will there be automated tests generated? - Can the model generate code that does error/exception handling correctly?
How (global). Questions about how the model operates to have a global understanding.	- How exactly does it generate code? - How does the translation enforce the dynamic typing rules of Python? - Can this AI use design patterns? - Does the AI optimize for big O?
Performance. Questions about the quality of AI generated artifacts or the runtime performance (e.g. time taken) of the model.	- How correct is the translation guaranteed to be? - How confident is the AI about the solution? - How will the performance be for code of larger inputs? - How much time will it take to translate the code?
How to. Questions about how to change the input to affect the quality or characteristics of the output	- Can I give hints or specify my problem better to improve the model's output? - How can I get the AI to write more efficient code? - How can I optimize for one type of output over another?
Control*. Questions about options to customize or specify preferences for how the model should work.	- Can I select a preference for [datatypes, packages, etc.] used? - Can I tune the translation? - Can I set specific regional settings?
Why/Why not. Questions about why the model produced a given output or why an input failed to produce the desired output.	- Why didn't my input work? - How did the AI recognize the function from comments in this task? - Why did the AI think that its code satisfied the requirement? - Why did that 4th attempt with the least programming thinking give a good result?
Data. Questions about the characteristics and provenance of the the data on which the model was trained.	- What data was this trained on? - What kind of training set was the AI built with? - Where does the code data the AI was trained on come from?
System Requirements & Impact†. Questions about the requirements to use the system or its impact, to gauge the appropriate conditions of usage.	- What are the hardware requirements for using the system? - Can I use the system in [a closed source project, my development environment, etc]? - What is the energy consumption for using the model?
Limitations†. Questions about the limitations of the model's capabilities.	- What limits are there to the model's function? - What scenarios does it cover or not cover?
What if. Questions about what the output would be if the input changes or in hypothetical situations.	- What to return if the input is invalid? - What if I am translating from a language that does not allow overflow.

Table 2: Question categories that emerged in our workshops. We applied the definitions of prototypical user questions in XAI Question Bank by Liao et al. [48] (in the context of discriminative ML, mainly for decision-support) and "intelligibility types" in Lim and Dey [53] (for context-aware intelligent systems). Categories marked with a asterisk (∗) are categories only appeared in Lim and Dey [53] not Liao et al. [48]. Categories with dagger (†) represent new categories emerged in our GAI for code context. For each question category, we provide the definition we used for coding and a few example questions asked by participants.

question category and what we can learn about users' explainability needs with GenAI for code.

Input. [3] To understand what kind of inputs the model can take or should be given was the most prominent explainability needs of participants, making up about 16% of all questions. On one hand, participants wanted to have an *overview* of the types or scope of inputs the AI can work with. Some asked about the AI's ability to process types of programming languages, data types, algorithms, language versions, and so on. On the other hand, participants were eager to know how to *optimize their inputs* to produce better outputs. For example, participants in W4-CA raised the question *"Should Alex write smaller functions that are easier to name or extensive documentation for complex functions?"* and *"Can I describe the entire problem in a single well-written sentence? or will I always need to break down the problem into small natural language chunks"* asked by W6-NL2Code.[4] Participants' questions reflected their current mental models on how the GenAI works to process inputs, which were not necessarily accurate, suggesting that explanations may need to start from a high-level overview of how GenAI works in a specific code generation use case. It is interesting to note that this category is not a prominent need for decision-support AI using descriminative ML, since the input is often fixed or implicit. In contrast, the great variability and close coupling between input and output for GenAI made it a primary explainability needs to address.

Output. To understand what output the GenAI can produce is another frequent explainability need. Participants were mainly interested in the *characteristics* of the output code, and the *scope* of the output or system functions. Understanding the characteristics of the output can help users determine how to utilize the output, for example P5 from W1-CT commented: *"you cannot ask Python to be as efficient as Java, but at the same time, Python may be more convenient than Java in some specific situations..."*. Understanding the characteristics can also guide users to assess the quality of output, identify potential shortcomings or errors for further actions, as commented by P19 from W4-CA: *"if it's translating from a language like RUST that doesn't allow overflow, it's not possible into a language like Java allows it."*. Meanwhile, some participants asked about the scope of the output, as to discover what the GenAI can do, such as whether it can generate test cases or multiple candidates as alternatives.

How (global). Participants also recognized the importance of having a a global understanding of the GenAI. Many wanted to have a high-level understanding of *how the codes are generated*, such as *"how exactly does it generate code? is it pulling from sites or using GANs to generate new code"* from W3-CA and *"how the model comprehends the user input"* from W6-NL2Code. Some raised these

questions based on their expectations of the input and output of the model (e.g., *"Python is dynamically typed, Java is static, how do you enforce these typing rules?"* from W1-CT. Others inquired about how the model *deals with specific types of input*, such as *"How untagged text is used for analysis"* from W9-CA. These questions reflected interests in both a high-level description of how code generation works, and examining detailed model logic. For descriptive ML, global explanations often take the forms of showing how the model weighs different features or describing the general rules the predictions follow. How to communicate such a global view of model logic for GenAI is an open technical challenge.

Performance. Many participants were concerned about understanding the performance, i.e. how well the GenAI works for the use case. These questions are critical for users to develop appropriate trust and adopt the system. We found the questions are mainly concerned about the *overall performance*, *the quality of a specific generated code* and the *run-time efficiency* of the model (e.g., inference time of the model, if the AI supports multi-threading). It is interesting to note that "quality of code" may be multi-faceted as participants mentioned correctness, runtime complexity, space complexity, understandability, and testability. Many of these performance-related questions were asked to understand *performance differences and limitations* with regard to different types of input (e.g. larger code). The interest in run-time performance is a unique need emerged in the GenAI for code context.

How to. These questions inquired about how to change or improve the input to get a better output. Answers to these questions can help participants choose strategies to make better use of the AI, such as *"How to [make] requirement better [to improve the output]"?* from W9-CA and *"How can I get the AI to write efficient code?"* from W6-NL2Code.

Control. These questions were regarding options to customize or specify preferences for how the model should work. This category was not seen in Liao et al.'s work defining explainability needs for decision-support descriminative ML. Emergence of this type of question in our context suggests that users of GenAI for code are interested in having more control on the working of the model, as illustrated by P61 from W1-CT saying that *"Oftentimes, you're not using the tool for just basic examples. I guess more in depth, like, explanations of what you can do with the model would be appreciated."*

Why/Why Not (local). Different from asking *How (global)* questions to understand the overall process or logic of the model, a *Why (local)* question was asked to understand a specific output from the model. More often than not, the *Why* question was triggered by a surprising or suspicious output, such as *"Why did the AI think the code satisfied the requirement?"* at W7-NL2Code or *"How did the AI recognize the function from comments in this task?"* at W4-CA. Sometimes the *Why* question was asked in contrast to the expected output, and hence a *Why Not* (the expected output) question, such as *"Why didn't my input work?"* from W3-CA.

Data. Some participants brought up questions regarding the training data, especially regarding the provenance of the data, such as *"Where does the code the AI was trained on come from?"* from W3-CA. Echoing findings in Liao et al. [48], explanations on training

---

[3]We adopt a similar definition of Input in Lim and Dey [53], as in what kind of inputs the model can take to generate code. Input was mentioned in Liao et al. [48] as training data went into the model. We consider that kind of question under the category of "Data".

[4]Please see Table 1 for an explanation of the meaning of "CA" and "NL2Code". The prepended "W" number is the number of the workshop in the table. Also, we some of our transcribed comments came from the audio channel of workshop recordings, and we could not reliably determine who had spoken. Other comments recorded on sticky-notes could be attributed. Therefore, we report participants' comments in terms of which *workshop* they occurred in, and we do not always know which *participant* actually said them. We report participant IDs where we could determine them.

data can help users gauge the capability of the model and its proper usage, by the validity of the training data and their alignment with ones' own programming tasks. How to define alignment of training data in the context of GenAI, however, remains an open question.

System Requirement and impact. A new category emerged in our data that was not covered in prior works. This category is questions regarding requirements or impact of the system, which can help users gauge the appropriate conditions of usage. They were asked regarding the usage of the system instead of the underlying model. These questions were highly specific to the software engineer context, including compatible development environments, software and hardware requirements, dependencies, and so on.

Limitations. Another category unique in our data is questions explicitly asking about the limitations of the model or system, such as what kind of scenarios it cannot cover or limitations to the model's functions. These questions emerged likely due to the complexity of the input and output spaces of GenAI, which is less straightforward to comprehend than a decision-support AI system.

What if. Some asked about what the output would be given hypothetical changes to the input, which may further help participants understand how the AI makes decisions in a counterfactual manner, such as *"what to return if the input is invalid?"* from W5-CT.

To summarize, we identified 11 categories of explainability needs in GenAI for code use cases. Four of these categories did not appear in Liao et al. as prominent needs for descriminative ML used for decision-support: Input, Control, System Requirements & Impact and Limitations. The top five most frequent categories are Input, Output, How (global), Performance and How to. In Section 6, we will further discuss insights revealed about explainabilty needs that are unique to the GenAI technology and the novel use case of supporting code generation.

5 XAI FEATURES FOR GENAI FOR CODE

As discussed in Section 3.2, we proposed four types of XAI features for GenAI for code and created design probes to elicit ideation from participants: AI documentation, uncertainty indicator, attention visualizer, and social transparency. In this section we discuss participants' response to derive concrete design recommendations.

5.1 AI Documentation

AI documentation is advocated to increase the transparency and facilitate understanding of AI [35, 36, 40, 75, 76, 82], and is embodied in recently proposed features of AI Factsheets [8, 81] and Model Cards [64]. However, what documentation should include for generative AI models is under-investigated, let alone GenAI for code use cases. We are interested in *what categories of information about the model should be presented in AI documentation for GenAI for code*, and how they might differ from that of discriminative ML. We used a low-fidelity UI design of a factsheet with intentionally open-ended content, as shown in Figure 2a, and asked participants to brainstorm AI documentation categories that could help the user in the design scenario.

We summarize the categories identified from data analysis in Table 3, ranked by the frequencies of mentions by participants. In particular, we identified categories that surfaced in the context

of GenAI for code that were not discussed in previous works on AI FactSheets or Model Cards: *Examples and tutorials*, *Software engineering capabilities*, *Output code quality and utility*, *Supported languages and frameworks*, *Control* and *Deployment Requirement & Platform*.

Consistent with findings from the question elicitation exercise in Section 5, participants expressed strong interests in understanding the scopes of inputs and outputs of the system. *Examples and tutorials*, as the most frequently mentioned category, suggests that example-based explanations can be a fruitful area to explore for GenAI to help users understand the model inputs and outputs. Such interests are also expressed as requirements specific for software engineering tasks, including *Software engineering capabilities* and *Supported languages and frameworks*. Participants also wished to see information about *performance*, *data*, *control*, *system deployment requirement*, and *global model explanations*, suggesting that providing documentation could help address corresponding explainability needs found in Section 5.

Another interesting pattern is that participants requested categories of information regarding the generated code artefacts rather than the model itself (see the second column in Table 3). Many demanded to see metrics about the *code quality and utility*, which should be differentiated from model performance metrics such as accuracy. While the latter are commonly calculated with some ground-truth in a held-out test dataset, the former are concerned with characterizing the generated code and their utility for improving human productivity.

In general, documentation for GenAI for code should be transparent about the quality of the produced code from the software engineering perspective, and align with the professional culture and standards, such as communicating the license and regulatory requirements in both the training data acquisition and the usage of the produced code. Table 3 is not meant to serve as an exhaustive template for AI documentation of generative AI models for code. Instead, we hope that these discovered categories can inspire what to document for generative AI models for code.

5.2 Model Uncertainty

Inspired by previous work on model uncertainty of generative AI models for code [100] and communicating uncertainty of output for discriminative ML models [10, 26], we created the design probe in Figure 2b to communicate uncertainty for a line of AI generated code, indicated by a wavy underline. We asked participants to react to the feature and discuss *how to improve the feature, including follow-up interactions, to better support users' understanding and their usage of the AI system.*

Their suggested improvement and follow-up interactions can be categorized as *AI initiated* and *human initiated* [38, 74]. For the AI to take the initiative, participants suggested to see more information from the model to help them understand the output and edit if necessary, for example to include suggesting alternative outputs, providing uncertainty explanations, and giving more fine-grained uncertainty. We elaborate on each category below.

Alternative outputs. Participants wanted transparency about what other alternatives the AI model had considered as output. Some participants had low expectations of generated code and hoped to

Category	Applied to	Definition
Examples & Tutorials[†]	Models	Examples of input-output pairs generated by the model; Tutorials on how to use the model effectively, including what kinds of input can get high-quality outputs
Software Engineering Capabilities[*]	Models	Description of software engineering features or capabilities that the AI can support (e.g., version information, stress testing for large loads, dependency handling, data structure, kernel status, coding style)
Model Performance	Models	Technical evaluation metrics of the generative model, including accuracy, performance, performance change by types of input, CPU consumption, and model inference time
Output Code Quality and Utility[*†]	Outputs	Metrics characterizing the generated code, including correctness, lint errors, code efficiency, time complexity; Metrics reflecting the system's impact on human productivity, including estimated time savings in conducting programming tasks, estimated improvements to code quality, comparisons with other kinds of GenAI for code tools
Supported Languages & Frameworks[*]	Models	List of programming and/or human languages which the model is capable of understanding (e.g. as input) or producing (e.g. as output); List of programming frameworks or APIs which the model supports as input or output (e.g. React, Flask)
Data	Models	Information about what data the model was trained on, including its provenance and any applicable privacy policies, data usage guidelines, or code licences
Control [†]	Models	Description of customization options or other mechanisms for users to control the output of the model; Description of how the model can be fine-tuned for additional use cases
Deployment Requirements & Platform[*]	Models	Technical requirements for hosting the model, including: software dependencies, hardware requirements, cloud-hosting requirements, and supported IDE integration
Model Explanations	Models	Explanation of how the model operates (e.g. a basic description of how transformer models work or a visualization of model attention)
Usage Rights	Outputs	Information about usage restrictions and/or licensing terms for code produced by the model
Optimal & Poor Conditions	Models	The conditions under which the AI model performs well or performs more poorly than expected
Intended Usage	Models	The use cases supported by the model; Other potential use cases that might be implemented via fine-tuning

Table 3: Categories of AI documentation for GenAI for code models and their outputs. Categories are sorted in descending order, based on the frequency of mention by participants. Categories marked with a † or ∗ have not been identified by previous works in AI documentation. We postulate that categories marked with a dagger (†) emerged due to the context of *generative* models, and categories marked with an asterisk (∗) emerged due to the context of software engineering support.

gain insights from more options, as P63 from W3-CA stated that *"I voted for the multiple [version] candidates, which I think is just generally helpful. Even if the top generated candidate was not quite right maybe one of the top five will be. Or will be better for whatever the particular use cases."*. Some participants hoped to see alternatives with specific characteristics (e.g., better readability or optimization), as P80 from W6-NL2Code stated: *"Obviously if human knows about the time complexity and the optimization of the code, he can fetch a better one. And at present, we do not know whether [the] AI [is] trying to be more on the readable side or more on the optimization side."*

Uncertainty explanation. Participants desired to see *uncertainty explanation* from the model, on why a particular part of the code was highlighted or why the AI was uncertain. This is an underexplored area in XAI. Participants suggested that the sources for

uncertainty explanations could be what alternative options the model was struggling with (W8-NL2Code), or what goal or rationale the model follows (W6-NL2Code). We quote P71 from W6-NL2Code who proposed an interactive explanation to address this point:

"I've voted twice for an indication of what the model is trying to achieve, because I think that's very important. It could be very simple. It could be as simple as just reminding you exactly what natural language prompt this was derived from, so you can think to yourself: 'what was it trying to do?' Uh, based on what I wrote, cause I know what I wrote, and I know what that means and then I can rewrite it from there or it could be more complex. It could, you know, if there's a serious natural

language model behind this, maybe it's capable of generating speech back to me about what this code exactly does and then you can compare that yourself."

Another path to explain uncertainty suggested by participants was through *highlighting corresponding parts* that contributed to the uncertainty. The corresponding parts could be from input, previously generated lines, or training data.

Fine-grained uncertainty. Some participants wanted to see more than just a wavy line under outputs that are below an uncertainty threshold. For example, P71 from W6-NL2Code said: *"...if there is some confidence percentage that could be helpful to, you know, like Green Zone, Red zone and Yellow zone etc., to give an idea of the highest priority I need to fix this."* Related to the observation that code quality is multi-faceted, some participants wanted to understand the specific definition or aspect of uncertainty, such as whether it is regarding the correctness, time complexity, or runtime.

Another area of suggested interactions requires the human to take the initiative, including allowing human input to resolve uncertainty and supporting interactive testing for the uncertain regions, as discussed below.

Human input to resolve uncertainty. A frequently requested interaction was that the AI waits for human input to resolve uncertainty. This could be realized by the AI prompting the human with a variety of questions, including confirmation (W3-CA), clarification (e.g., *"show did you mean ... options"* from W1-CT), inspection (e.g., *"Tell user to examine the line carefully and verify"* from W4-CA), preferences or how they would code (e.g., *"Ask user for decision making – 'do you want to keep this line or not' and highlight it with reasoning"* from W5-CT), and immediate modification (e.g., *"Dialog box to type preferred translation (if Alex knows python")* from W2-CT)).

Interactive testing. Another interaction that participants expected to perform is interactive testing for situations in which the model was uncertain, which can also facilitate understanding of the AI model. P81 from W6-NL2Code said that *"... If we can probably give a similar interactive component [as Jupyter Notebook] where it gives you the option to fetch a context from the previous clients, and then run only that piece of code to see what the output is. Probably the person can make a better decision on whether to keep it or change."*

The overall feedback from participants was that showing a line-level uncertainty indicator would be useful to guide users' attention to potentially low-quality code. However, showing uncertainty alone was not enough to satisfy their explainability needs. Many of the suggested interactions also serve the purpose of helping users better understand the model behaviors through interactive testing, alternative output it considered, and explaining the uncertainty. Through these interactions, users can develop a more appropriate mental model of the system that can help them interact more effectively in the long run.

5.3 Attention Distribution

Attention distribution is a popular approach to explain an individual prediction made by a neural networks model, by the relative importance of parts of input to determine the output [95, 96]. Mapping to the context of GenAI for code, we designed the UI in Figure 2c,

where the user can select a span of generated content. The AI can then highlight some spans in the previous content, to illustrate the idea of attention when making local decisions, indicating different attention weights via different opacity. We showed it to participants to gather their response on *the utility of attention distribution visualizer for users of GenAI for code.*

In general, participants said that the attention features would be useful to help them understand how the model worked, and to guide them in modifying the input to produce better results. P42 from W7-NL2Code said that *" It's what trains me as the user to use the AI better"* and P63 from W3-CA said that *"if you see an issue, you might be able to say, oh, I can go back and try tweaking this part of the input and see whether I can get a better result. So that could be useful."*

Participants also proposed potential ways to improve this local explanation feature. For example: 1) line-level interpretation might be more appropriate than the span level, as P53 in W3-CA said: *"selecting a span may not be the best option for a user interface; potentially a line is better"*; 2) being able to make immediate changes *"right after identifying the suspicious spots"* via such a tool (P55 in W4-CA); 3) syntax tree driven selection so that *"if you click on the range, you should get the whole function not only the span"* (P63 in W3-CA); 4) adding natural language explanation in addition to the color coding (P71 in W6-NL2Code and P22 in W8-NL2Code).

Moreover, participants saw value in utilizing user edits or interaction with attention distribution as signals to improve the model or future interactions, for example, collecting the user edits, as P63 in W3-CA said that *"it might be more useful for how do we improve the model? Cause if somebody... starts typing in, here's the code that I really wanted here. And you can see how does that align and say, wait, why isn't this aligning? It might be more useful is user feedback back to the model builders."*

5.4 Social Transparency

Social transparency (ST) in AI system is recently proposed by Ehsan et al. [19], built on the concept of social transparency in social computing[90]. ST makes visible socio-organizational factors that influence the use of AI by presenting other users' interactions with the AI, aiming to facilitate a better understanding of and effective actions with AI output. To explore ST in the context of GenAI for code, we introduced an open-ended probe in Figure 2d: an image highlighting that Alex was not working alone but together with a team of software engineers, all using the GenAI. We asked participants to ideate on *what users might want to know about other team members' interactions with the AI to help them better understand and use the AI.* We asked participants to write down their ideas in Mural, and then did the clustering, voting and sharing again. We collected 111 responses about ST from Mural, grouped them to categories, and sorted them based on the frequencies of their occurrences. With the open-ended probe, we were able to elicit user needs for ST in the broad context of software engineer work with the participation of GenAI for code. We therefore map the findings to a stage model of the software development life cycle based on Rani [79]: 1) Requirement Analysis; 2) Design; 3) Implementation; 4) Testing; 5) Deployment and Maintenance [79].

At the stage of requirement analysis, participants wanted to know about the *team information* of who, and for what, will be using the AI, such as business requirements and goals of the team, information about each team member (e.g., programming skills and experience), their interaction patterns with the AI (e.g., who is frequently using AI and who is better at using AI) and the reason why they are using the AI. Knowing the team information early on could help users better coordinate and interpret ST information in later stages.

Information about other people's *experience with the AI* could help at the design stage so that users could better design their programming tasks with the GenAI, including taking precautions on when and how they need to take more control on the code generation. It is especially helpful to learn about the AI's performance, capability and limitations from others' experience at this stage. For example, participants wanted to know how other people had evaluated the performance of the model and their judgement of the code quality, including how much time they invested in getting started with AI, how the AI-generated code compared to human-written code, together with the strengths and weaknesses of the AI. Moreover, they were interested in seeing limitations, such as common pitfalls and errors that other members had experienced.

Going into the development stage, participants wanted to have information about *similar tasks or requests* of other people to better understand what the AI model was capable of doing, even re-use the generated content, to help them better manage the programming tasks. Some participants expressed a desire to always have clarity on the *authoring entity* of code (whether from AI, a colleague, or other resources) throughout the development stage to better manage their expectations, actions as well as accountability.

At the testing stage, participants wished to know how other people had tested the AI model as a reference, including what measures they had used in their evaluation of code quality. Many requested to understand AI's *impact on humans* at this stage, i.e. how the introduction of the AI influenced the productivity of the team members, such as team speed and productivity. They also wanted to know whether and how the AI-generated code might be added to the codebase.

Finally, during the deployment and maintenance stage, participants asked for information about how to customize the model parameters and monitored metrics based on the preferences of the team, and how to collect the team's edits and feedback to improve the model.

These results move beyond Ehsan et al. [19]'s work that defined ST features at the point of AI decision-support. Our results open up the design space to embed rich ST features at different stages of software development. ST features in the context of GenAI for code can not only facilitate users to better understand and use the AI, but also potentially foster a more effective "human-AI assemblage" where GenAI is introduced as a member or assistant of a software engineer team. We encourage future work to further explore the design space.

6 DISCUSSION

6.1 Informing XAI approaches for GenAI for code

The rich and complex inputs and outputs of GenAI, and the tight coupling of the two, make it a prominent need for users to form a clear mental model on the scope, characteristics, and limitations of the input and output spaces, as well as a global view of how the model generates outputs from inputs. The observation that participants' explainability needs focused overwhelmingly on input, output and global explanations, suggests a mismatch between what users of GenAI applications need and the technical community's current focus on explanations for representation learning and visualizing representations.

The explainability categories we identified have varied technical feasibility with current techniques, and point to topics that are under-explored for generative AI. For example, for the *Performance* category, existing works have used the Computational Accuracy metric to evaluate generative code models [9, 15, 33, 88], but not other metrics we uncovered regarding the characteristics of the generated artifacts and run-time efficiency. To understand performance differences and limitations with regard to different types of input, solutions have been explored for natural language generation under Prompt Engineering [57, 58]. Similar studies in the code space are limited to the number of few-shot examples in the prompt, and the effect of length of prompt on performance [9, 15]. A more detailed analysis of these phenomenon, better grounded in the semantics of programming languages, could be a promising area for future research. For the *Control* category, it is feasible to directly interact with model parameters. However, questions of how to elicit human feedback or edits and incorporate them to train generative models are currently under explored.

XAI for GenAI needs to be contextualized by the characteristics of the generated artifacts, and the practical and cultural contexts to use these artifacts, i.e. the software engineering domain in our case. For example, participants were interested in understanding the input and output spaces with regard to supported language, frameworks, data structures, and so on. They are also interested in metrics reflecting various aspects of code characteristics and impact on human productivity rather than technical accuracy alone.

Participants' questions reflected their desire for an *actionable* or *utility-oriented* understanding to support their end goal of optimizing code generation and programming productivity [18, 50, 52], such as asking the *Input, Output, How* and *How-to* XAI questions (described in Section 4) to facilitate strategies to get better outputs from the AI. This actionable understanding can also be supported by enabling follow-up actions towards their goals after seeing transparent information. For example, participants suggested many interactions to allow them to act on uncertainty information and improve the generated code.

In short, explainability needs should be understood and addressed with the ecological factors in mind, situated in an understanding of the behavioral and social contexts of the system [41, 70]. We paid attention to the nature of software engineer tasks and workflows as part of the pragmatic human environment of the system. For example, participants demanded information regarding system requirements and impacts in relation to their broader work

environment. The discussions around ST further suggested that users' explainability needs may vary at different stages of software engineering lifecycle, adding *temporal* context to the work contexts. We encourage future research to consider these contextual aspects while designing and evaluating GenAI explanations.

6.2 Design implications for GenAI for code

In our study, we observed a positive user reception towards natural language explanations and interactions with the GenAI as user inputs are programming- or natural-language based. We envision that a conversational agent interface could be a natural fit for AI assistant for code or co-programming tools, as explored in recent work [42, 43]

We also noticed that the explainability needs and user needs in general may differ for users with varied levels of programming skills. Less experienced participants generally asked fewer questions in the workshop. It is possible that they face more challenges articulating or realizing they had certain explainability needs, and may benefit from more proactive explanations or an entire interaction session focusing on training and explaining rather than in-situ explanations. Future research should further explore user needs in GenAI for code use cases that target reducing programming barriers and enabling novices to code.

Lastly, the bulk of explainability needs around GenAI for code leads to the question of utility of GenAI for code itself. One concern stems from the assumption that users are burdened with understanding how the AI works and adjusting their inputs for optimal outcomes, as expressed by P7 in W8-NL2Code: *"as you were talking about this in this session, I keep thinking, if you are learning about all the use cases and how you're going to structure the natural language and everything, how is it kind of different than just learning an actual language?".* We also note that at the current time, generative code models are prone to errors and will require post-generation improvements from humans [69, 100]. There is a fundamental question on the readiness of the technology, which may not be able to addressed by providing explainability alone. We urge the AI and HCI community to evaluate GenAI for code technologies, define intended use, and refrain from use cases that have not been validated or have risks of harmful consequences for stakeholders.

6.3 Human-centered, participatory, and question-driven approaches to XAI design

Our study used a scenario-based design [87] combined with a question-driven method to elicit explaiinability needs, based on work by Liao et al. [48, 50]. We reflect on a few strategies that worked well for our study, and encourage researchers and practitioners to adopt similar approaches to understand users' explainability needs early-on to drive technical and design choices.

First, we found the use of a scenario, persona and an illustrated prototype corresponding to the scenario to effectively aid the elicitation of rich user questions, even for a technology that was completely new for participants. As Liao et al. [50] suggested "for highly novel systems, scenarios or low-fi prototypes can be used to elicit questions". Moreover, we incorporated real outputs from state-of-the-art generative code models in the low-fi prototype to illustrate

the model capabilities, which was repeatedly inquired by participants to confirm the scenario has a realistic reflection of the GenAI capabilities. This approach also echos a recent trend in AI design to utilize real data points as "data probes" to aid design ideation [91]. The choice of data point can sway the discussions in the user study. We suggest to choose ones that reflect the AI's true capabilities, including limitations and errors to explore the design space more thoroughly.

Second, we found the procedure of "pooling-clustering-voting" questions to be productive. While Liao et al. [50]'s original method only deals with question elicitation in one-on-one interviews, we adopted a participatory workshop format to balance between individual brainstorming and group discussions. Giving participants seven minutes to post their questions allowed a quantity of questions to be added. Working collaboratively to cluster them and vote on the clusters encouraged discussions and building upon each other's ideas. This procedure also allowed participants to naturally articulate reasons behind their questions for our data collection.

Lastly, in addition to the question elicitation exercise, we explored four types of XAI features with low-fi prototypes. We defined these features based on prior works in different technical and application domains. A critical study design decision we had to make is the open-endedness of these prototypes. For some features (uncertainty indicators and attention visualizers), there already exist relevant techniques, so we chose a more concrete design to elicit feedback and ideas for extension. For other features (documentation and ST), the design space is less defined so we used more open-ended probes to allow participants to freely and critically brainstorm. In short, the design of a "probe" for participatory ideation and feedback is a non-trivial design decision, requiring a balance between concreteness and openness, considerations of both technical feasibility and user value, and diligent pilot testing [12, 24].

7 CONCLUSION

Despite growing efforts to apply state-of-the-art GenAI models to support software engineering tasks, investigations on user needs for such technologies have been scarce. Our work is among the first to study users' explainability needs of GenAI for code. By combining scenario-based design [87] and a recently proposed question-driven design method for XAI [48, 50], we conducted 9 participatory workshops with 43 software engineers to understand their explainability needs with three use cases of GenAI for code: natural language to code, code translation, and code auto-completion. As a result, we identified 11 categories of explainability needs in the context of GenAI for code. We provided detailed definitions and examples for these categories, and contrasted them with common explainability needs for discriminative ML discussed in prior work [48, 53]. In addition, we proposed four areas of XAI features for these use cases, collected feedback from participants and provided concrete design recommendations. We hope that our results can inspire future AI and HCI work that can enable better human-AI collaboration for software engineering, and encourage more human-centered approaches to drive AI technical development.

REFERENCES

[1] Amina Adadi and Mohammed Berrada. 2018. Peeking inside the black-box: A survey on Explainable Artificial Intelligence (XAI). *IEEE Access* 6 (2018), 52138–52160.

[2] Mayank Agarwal, Kartik Talamadupula, Stephanie Houde, Fernando Martinez, Michael J. Muller, John T. Richards, Steven Ross, and Justin D. Weisz. 2020. Quality Estimation & Interpretability for Code Translation. *ArXiv* abs/2012.07581 (2020).

[3] Wasi Ahmad, Saikat Chakraborty, Baishakhi Ray, and Kai-Wei Chang. 2021. Unified Pre-training for Program Understanding and Generation. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies.* Association for Computational Linguistics, Online, 2655–2668. https://doi.org/10.18653/v1/2021.naacl-main.211

[4] Miltiadis Allamanis, Earl T Barr, Premkumar Devanbu, and Charles Sutton. 2018. A survey of machine learning for big code and naturalness. *ACM Computing Surveys (CSUR)* 51, 4 (2018), 1–37.

[5] Saleema Amershi, Maya Cakmak, William Bradley Knox, and Todd Kulesza. 2014. Power to the people: The role of humans in interactive machine learning. *Ai Magazine* 35, 4 (2014), 105–120.

[6] Cecilia Aragon, Shion Guha, Marina Kogan, Michael Muller, and Gina Neff. 2022. *Human-Centered Data Science: An Introduction.* MIT Press, Cambridge, MA.

[7] Cecilia Aragon, Clayton Hutto, Andy Echenique, Brittany Fiore-Gartland, Yun Huang, Jinyoung Kim, Gina Neff, Wanli Xing, and Joseph Bayer. 2016. Developing a research agenda for human-centered data science. In *Proceedings of the 19th ACM Conference on Computer Supported Cooperative Work and Social Computing Companion.* 529–535.

[8] Matthew Arnold, Rachel KE Bellamy, Michael Hind, Stephanie Houde, Sameep Mehta, Aleksandra Mojsilović, Ravi Nair, K Natesan Ramamurthy, Alexandra Olteanu, David Piorkowski, et al. 2019. FactSheets: Increasing trust in AI services through supplier's declarations of conformity. *IBM Journal of Research and Development* 63, 4/5 (2019), 6–1.

[9] Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, et al. 2021. Program Synthesis with Large Language Models. *arXiv preprint arXiv:2108.07732* (2021).

[10] Umang Bhatt, Javier Antorán, Yunfeng Zhang, Q Vera Liao, Prasanna Sattigeri, Riccardo Fogliato, Gabrielle Melançon, Ranganath Krishnan, Jason Stanley, Omesh Tickoo, et al. 2021. Uncertainty as a form of transparency: Measuring, communicating, and using uncertainty. In *Proceedings of the 2021 AAAI/ACM Conference on AI, Ethics, and Society.* 401–413.

[11] Umang Bhatt, Alice Xiang, Shubham Sharma, Adrian Weller, Ankur Taly, Yunhan Jia, Joydeep Ghosh, Ruchir Puri, José MF Moura, and Peter Eckersley. 2020. Explainable machine learning in deployment. In *Proceedings of the 2020 Conference on Fairness, Accountability, and Transparency.* 648–657.

[12] Kirsten Boehner, Janet Vertesi, Phoebe Sengers, and Paul Dourish. 2007. How HCI interprets the probes. In *Proceedings of the SIGCHI conference on Human factors in computing systems.* 1077–1086.

[13] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, T. J. Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeff Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language Models are Few-Shot Learners. *ArXiv* abs/2005.14165 (2020).

[14] Rich Caruana, Yin Lou, Johannes Gehrke, Paul Koch, Marc Sturm, and Noemie Elhadad. 2015. Intelligible models for healthcare: Predicting pneumonia risk and hospital 30-day readmission. In *Proceedings of KDD.*

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

[16] Tian Qi Chen, Xuechen Li, Roger B. Grosse, and David Kristjanson Duvenaud. 2018. Isolating Sources of Disentanglement in Variational Autoencoders. In *NeurIPS.*

[17] Premkumar Devanbu. 2015. New initiative: The naturalness of software. In *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*, Vol. 2. IEEE, 543–546.

[18] Shipi Dhanorkar, Christine T Wolf, Kun Qian, Anbang Xu, Lucian Popa, and Yunyao Li. 2021. Who needs to know what, when?: Broadening the Explainable AI (XAI) Design Space by Looking at Explanations Across the AI Lifecycle. In *Designing Interactive Systems Conference 2021.* 1591–1602.

[19] Upol Ehsan, Q Vera Liao, Michael Muller, Mark O Riedl, and Justin D Weisz. 2021. Expanding explainability: Towards social transparency in ai systems. In *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems.* 1–19.

[20] Upol Ehsan and Mark Riedl. 2021. *Explainability Pitfalls: Beyond Dark Patterns in Explainable AI - paper at HCAI@NeurIPS2021 workshop on human centered AI.* https://sites.google.com/view/hcai-human-centered-ai-neurips/home Accessed January 19, 2022.

[21] Upol Ehsan and Mark O Riedl. 2020. Human-centered explainable ai: Towards a reflective sociotechnical approach. In *International Conference on Human-Computer Interaction.* Springer, 449–466.

[22] Upol Ehsan, Philipp Wintersberger, Q Vera Liao, Martina Mara, Marc Streit, Sandra Wachter, Andreas Riener, and Mark O Riedl. 2021. Operationalizing Human-Centered Perspectives in Explainable AI. In *Extended Abstracts of the 2021 CHI Conference on Human Factors in Computing Systems.* 1–6.

[23] Zhangyin Feng, Daya Guo, Duyu Tang, Nan Duan, Xiaocheng Feng, Ming Gong, Linjun Shou, Bing Qin, Ting Liu, Daxin Jiang, et al. 2020. Codebert: A pre-trained model for programming and natural languages. *arXiv preprint arXiv:2002.08155* (2020).

[24] Bill Gaver, Tony Dunne, and Elena Pacenti. 1999. Design: Cultural Probes. *Interactions* 6, 1 (jan 1999), 21–29. https://doi.org/10.1145/291224.291235

[25] Werner Geyer, Lydia B Chilton, Justin D Weisz, and Mary Lou Maher. 2021. HAI-GEN 2021: 2nd Workshop on Human-AI Co-Creation with Generative Models. In *26th International Conference on Intelligent User Interfaces.* 15–17.

[26] Soumya Ghosh, Q Vera Liao, Karthikeyan Natesan Ramamurthy, Jiri Navratil, Prasanna Sattigeri, Kush R Varshney, and Yunfeng Zhang. 2021. Uncertainty Quantification 360: A Holistic Toolkit for Quantifying and Communicating the Uncertainty of AI. *arXiv preprint arXiv:2106.01410* (2021).

[27] Github. 2021. Copilot. Retrieved 03-August-2021 from https://copilot.github.com

[28] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. 2020. Generative adversarial networks. *Commun. ACM* 63, 11 (2020), 139–144.

[29] Riccardo Guidotti, Anna Monreale, Salvatore Ruggieri, Franco Turini, Fosca Giannotti, and Dino Pedreschi. 2018. A survey of methods for explaining black box models. *ACM computing surveys (CSUR)* 51, 5 (2018), 1–42.

[30] Daya Guo, Shuo Ren, Shuai Lu, Zhangyin Feng, Duyu Tang, Shujie Liu, Long Zhou, Nan Duan, Alexey Svyatkovskiy, Shengyu Fu, et al. 2020. Graphcodebert: Pre-training code representations with data flow. *arXiv preprint arXiv:2009.08366* (2020).

[31] Shunan Guo, Fan Du, Sana Malik, Eunyee Koh, Sungchul Kim, Zhicheng Liu, Donghyun Kim, Hongyuan Zha, and Nan Cao. 2019. Visualizing uncertainty and alternatives in event sequence predictions. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems.* 1–12.

[32] Aaron Halfaker and R Stuart Geiger. 2020. Ores: Lowering barriers with participatory machine learning in wikipedia. *Proceedings of the ACM on Human-Computer Interaction* 4, CSCW2 (2020), 1–37.

[33] Dan Hendrycks, Steven Basart, Saurav Kadavath, Mantas Mazeika, Akul Arora, Ethan Guo, Collin Burns, Samir Puranik, Horace He, Dawn Song, et al. 2021. Measuring Coding Challenge Competence With APPS. *arXiv preprint arXiv:2105.09938* (2021).

[34] Denis J Hilton. 1990. Conversational processes and causal explanation. *Psychological Bulletin* 107, 1 (1990), 65.

[35] M. Hind, Stephanie Houde, Jacquelyn Martino, A. Mojsilovic, David Piorkowski, John T. Richards, and K. Varshney. 2020. Experiences with Improving the Transparency of AI Models and Services. *Extended Abstracts of the 2020 CHI Conference on Human Factors in Computing Systems* (2020).

[36] M. Hind, S. Mehta, A. Mojsilovic, R. Nair, K. Ramamurthy, Alexandra Olteanu, and K. Varshney. 2019. Increasing Trust in AI Services through Supplier's Declarations of Conformity. *IBM J. Res. Dev.* 63 (2019), 6:1–6:13.

[37] Abram Hindle, Earl T Barr, Mark Gabel, Zhendong Su, and Premkumar Devanbu. 2016. On the naturalness of software. *Commun. ACM* 59, 5 (2016), 122–131.

[38] Eric Horvitz. 1999. Principles of Mixed-Initiative User Interfaces. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (Pittsburgh, Pennsylvania, USA) *(CHI '99).* Association for Computing Machinery, New York, NY, USA, 159–166. https://doi.org/10.1145/302979.303030

[39] Seohyun Kim, Jinman Zhao, Yuchi Tian, and Satish Chandra. 2021. Code prediction by feeding trees to transformers. In *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE).* IEEE, 150–162.

[40] Bran Knowles and John T. Richards. 2021. The Sanction of Authority: Promoting Public Trust in AI. *Proceedings of the 2021 ACM Conference on Fairness, Accountability, and Transparency* (2021).

[41] Marina Kogan, Aaron Halfaker, Shion Guha, Cecilia Aragon, Michael Muller, and Stuart Geiger. 2020. Mapping Out Human-Centered Data Science: Methods,

Approaches, and Best Practices. In *Companion of the 2020 ACM International Conference on Supporting Group Work*. 151–156.

[42] Sandeep Kaur Kuttal, Jarow Myers, Sam Gurka, David Magar, David Piorkowski, and Rachel Bellamy. 2020. Towards designing conversational agents for pair programming: Accounting for creativity strategies and conversational styles. In *2020 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*. IEEE, 1–11.

[43] Sandeep Kaur Kuttal, Bali Ong, Kate Kwasny, and Peter Robe. 2021. Trade-offs for Substituting a Human with an Agent in a Pair Programming Context: The Good, the Bad, and the Ugly. In *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems*. 1–20.

[44] Himabindu Lakkaraju, Stephen H. Bach, and Jure Leskovec. 2016. Interpretable Decision Sets: A Joint Framework for Description and Prediction. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (San Francisco, California, USA) *(KDD '16)*. Association for Computing Machinery, New York, NY, USA, 1675–1684. https://doi.org/10.1145/2939672.2939874

[45] Min Kyung Lee, Nina Grgić-Hlača, Michael Carl Tschantz, Reuben Binns, Adrian Weller, Michelle Carney, and Kori Inkpen. 2020. Human-centered approaches to fair and responsible AI. In *Extended Abstracts of the 2020 CHI Conference on Human Factors in Computing Systems*. 1–8.

[46] Min Kyung Lee, Daniel Kusbit, Anson Kahng, Ji Tae Kim, Xinran Yuan, Allissa Chan, Daniel See, Ritesh Noothigattu, Siheon Lee, Alexandros Psomas, et al. 2019. WeBuildAI: Participatory framework for algorithmic governance. *Proceedings of the ACM on Human-Computer Interaction* 3, CSCW (2019), 1–35.

[47] Tao Lei, Regina Barzilay, and T. Jaakkola. 2016. Rationalizing Neural Predictions. In *EMNLP*.

[48] Q Vera Liao, Daniel Gruen, and Sarah Miller. 2020. Questioning the AI: informing design practices for explainable AI user experiences. In *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems*. 1–15.

[49] Q Vera Liao and Michael Muller. 2019. Enabling Value Sensitive AI Systems through Participatory Design Fictions. *arXiv preprint arXiv:1912.07381* (2019).

[50] Q Vera Liao, Milena Pribić, Jaesik Han, Sarah Miller, and Daby Sow. 2021. Question-Driven Design Process for Explainable AI User Experiences. *arXiv preprint arXiv:2104.03483* (2021).

[51] Q Vera Liao, Moninder Singh, Yunfeng Zhang, and Rachel Bellamy. 2021. Introduction to explainable ai. In *Extended Abstracts of the 2021 CHI Conference on Human Factors in Computing Systems*. 1–3.

[52] Q Vera Liao and Kush R Varshney. 2021. Human-Centered Explainable AI (XAI): From Algorithms to User Experiences. *arXiv preprint arXiv:2110.10790* (2021).

[53] Brian Y Lim and Anind K Dey. 2010. Toolkit to support intelligibility in context-aware applications. In *Proceedings of the 12th ACM international conference on Ubiquitous computing*. 13–22.

[54] Brian Y Lim, Anind K Dey, and Daniel Avrahami. 2009. Why and why not explanations improve the intelligibility of context-aware intelligent systems. In *Proceedings of the SIGCHI conference on human factors in computing systems*. 2119–2128.

[55] Pantelis Linardatos, Vasilis Papastefanopoulos, and Sotiris B. Kotsiantis. 2021. Explainable AI: A Review of Machine Learning Interpretability Methods. *Entropy* 23 (2021).

[56] Zachary C Lipton. 2018. The mythos of model interpretability. *Queue* 16, 3 (2018), 31–57.

[57] Jiachang Liu, Dinghan Shen, Yizhe Zhang, Bill Dolan, Lawrence Carin, and Weizhu Chen. 2021. What Makes Good In-Context Examples for GPT-3? *arXiv preprint arXiv:2101.06804* (2021).

[58] Pengfei Liu, Weizhe Yuan, Jinlan Fu, Zhengbao Jiang, Hiroaki Hayashi, and Graham Neubig. 2021. Pre-train, prompt, and predict: A systematic survey of prompting methods in natural language processing. *arXiv preprint arXiv:2107.13586* (2021).

[59] Ryan Louie, Andy Coenen, Cheng Zhi Huang, Michael Terry, and Carrie J Cai. 2020. Novice-AI music co-creation via AI-steering tools for deep generative models. In *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems*. 1–13.

[60] Ryan Louie, Any Cohen, Cheng-Zhi Anna Huang, Michael Terry, and Carrie J Cai. 2020. Cococo: AI-Steering Tools for Music Novices Co-Creating with Generative Models.. In *HAI-GEN+ user2agent@ IUI*.

[61] Shuai Lu, Daya Guo, Shuo Ren, Junjie Huang, Alexey Svyatkovskiy, Ambrosio Blanco, Colin Clement, Dawn Drain, Daxin Jiang, Duyu Tang, Ge Li, Lidong Zhou, Linjun Shou, Long Zhou, Michele Tufano, Ming Gong, Ming Zhou, Nan Duan, Neel Sundaresan, Shao Kun Deng, Shengyu Fu, and Shujie Liu. 2021. CodeXGLUE: A Machine Learning Benchmark Dataset for Code Understanding and Generation. *ArXiv abs/2102.04664* (2021).

[62] Scott M Lundberg and Su-In Lee. 2017. A unified approach to interpreting model predictions. In *Proceedings of the 31st international conference on neural information processing systems*. 4768–4777.

[63] Cade Metz. 2021. A.I. Can Now Write Its Own Computer Code. That's Good News for Humans. *The New York Times* (9 September 2021). https://www.nytimes.com/2021/09/09/technology/codex-artificial-intelligence-coding.html

[64] 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. *Proceedings of the Conference on Fairness, Accountability, and Transparency* (2019).

[65] Michael Muller, Plamen Angelov, Shion Guha, Marina Kogan, Gina Neff, Nuria Oliver, Manuel Gomez Rodriquez, and Adrian Weller. 2021. *HCAI@NeurIPS2021: Human Centered AI workshop at NeurIPS 2021.* https://sites.google.com/view/hcai-human-centered-ai-neurips/home Accessed January 17, 2022.

[66] Michael Muller, Cecilia Aragon, Shion Guha, Marina Kogan, Gina Neff, Cathrine Seidelin, Katie Shilton, and Anissa Tanweer. 2020. Interrogating Data Science. In *Conference Companion Publication of the 2020 on Computer Supported Cooperative Work and Social Computing*. 467–473.

[67] Michael Muller, Melanie Feinberg, Timothy George, Steven J Jackson, Bonnie E John, Mary Beth Kery, and Samir Passi. 2019. Human-centered study of data science work practices. In *Extended Abstracts of the 2019 CHI Conference on Human Factors in Computing Systems*. 1–8.

[68] Michael Muller and Q Vera Liao. [n.d.]. Exploring AI Ethics and Values through Participatory Design Fictions. ([n. d.]).

[69] Michael Muller, April Y. Wang, Steven I. Ross, Justin D. Weisz, Mayank Agarwal, Kartik Talamadupula, Stephanie Houde, Fernando Martinez, John Richards, Jaimie Drozdal, Xie Lui, David Piorkowski, and Dakuo Wang. 2021. *How data scientists improve generated code documentation in Jupyter notebooks.* Retrieved October 5, 2021 from https://hai-gen2021.github.io/program/

[70] Michael Muller, Christine T Wolf, Josh Andres, Michael Desmond, Narendra Nath Joshi, Zahra Ashktorab, Aabhas Sharma, Kristina Brimijoin, Qian Pan, Evelyn Duesterwald, et al. 2021. Designing Ground Truth and the Social Life of Labels. In *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems*. 1–16.

[71] Anh Tuan Nguyen, Tung Thanh Nguyen, and Tien N Nguyen. 2014. Migrating code with statistical machine translation. In *Companion Proceedings of the 36th International Conference on Software Engineering*. 544–547.

[72] Yusuke Oda, Hiroyuki Fudaba, Graham Neubig, Hideaki Hata, Sakriani Sakti, Tomoki Toda, and Satoshi Nakamura. 2015. Learning to generate pseudo-code from source code using statistical machine translation (t). In *2015 30th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 574–584.

[73] Andrés Páez. 2019. The pragmatic turn in explainable artificial intelligence (XAI). *Minds and Machines* 29, 3 (2019), 441–459.

[74] Raja Parasuraman, Thomas B Sheridan, and Christopher D Wickens. 2000. A model for types and levels of human interaction with automation. *IEEE Transactions on systems, man, and cybernetics-Part A: Systems and Humans* 30, 3 (2000), 286–297.

[75] David Piorkowski, D. Gonz'alez, John T. Richards, and Stephanie Houde. 2020. Towards evaluating and eliciting high-quality documentation for intelligent systems. *ArXiv abs/2011.08774* (2020).

[76] David Piorkowski, Soya Park, A. Wang, Dakuo Wang, Michael J. Muller, and Felix Portnoy. 2021. How AI Developers Overcome Communication Challenges in a Multidisciplinary Team. *Proceedings of the ACM on Human-Computer Interaction* 5 (2021), 1 – 25.

[77] Ruchi Puri, D. Kung, G. Janssen, Wei Zhang, Giacomo Domeniconi, Vladmir Zolotov, Julian Dolby, Jie Chen, M. Choudhury, Lindsey Decker, Veronika Thost, Luca Buratti, Saurabh Pujar, and Ulrich Finkler. 2021. Project CodeNet: A Large-Scale AI for Code Dataset for Learning a Diversity of Coding Tasks. *ArXiv abs/2105.12655* (2021).

[78] Inioluwa Deborah Raji and Jingying Yang. 2019. About ml: Annotation and benchmarking on understanding and transparency of machine learning lifecycles. *arXiv preprint arXiv:1912.06166* (2019).

[79] Sahil Barjtya Ankur Sharma Usha Rani. 2017. A detailed study of Software Development Life Cycle (SDLC) Models. *International Journal of Engineering and Computer Science* 6 (2017).

[80] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. 2016. Why should i trust you?: Explaining the predictions of any classifier. In *Proceedings of KDD*.

[81] John Richards, David Piorkowski, Michael Hind, Stephanie Houde, and Aleksandra Mojsilović. 2020. A Methodology for Creating AI FactSheets. *arXiv preprint arXiv:2006.13796* (2020).

[82] John T. Richards, David Piorkowski, M. Hind, Stephanie Houde, and Aleksandra Mojsilovi'c. 2020. A Methodology for Creating AI FactSheets. *ArXiv abs/2006.13796* (2020).

[83] Karl Ridgeway. 2016. A Survey of Inductive Biases for Factorial Representation-Learning. *ArXiv abs/1612.05299* (2016).

[84] Karl Ridgeway and Michael C. Mozer. 2018. Learning Deep Disentangled Embeddings with the F-Statistic Loss. In *NeurIPS*.

[85] Mark O Riedl. 2019. Human-centered artificial intelligence and machine learning. *Human Behavior and Emerging Technologies* 1, 1 (2019), 33–36.

[86] Andrew Ross, Nina Chen, Elisa Zhao Hang, Elena L Glassman, and Finale Doshi-Velez. 2021. Evaluating the Interpretability of Generative Models by Interactive Reconstruction. In *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems*. 1–15.

[87] Mary Beth Rosson and John M Carroll. 2009. Scenario-based design. In *Human-computer interaction*. CRC Press, 161–180.

[88] Baptiste Roziere, Marie-Anne Lachaux, Lowik Chanussot, and Guillaume Lample. 2020. Unsupervised Translation of Programming Languages.. In *NeurIPS*.

[89] Ben Shneiderman. 2020. Bridging the gap between ethics and practice: Guidelines for reliable, safe, and trustworthy Human-Centered AI systems. *ACM Transactions on Interactive Intelligent Systems (TiiS)* 10, 4 (2020), 1–31.

[90] H Colleen Stuart, Laura Dabbish, Sara Kiesler, Peter Kinnaird, and Ruogu Kang. 2012. Social transparency in networked information exchange: a theoretical framework. In *Proceedings of the ACM 2012 conference on Computer Supported Cooperative Work*. 451–460.

[91] Hariharan Subramonyam, Colleen Seifert, and Eytan Adar. 2021. Towards A Process Model for Co-Creating AI Experiences. *arXiv preprint arXiv:2104.07595* (2021).

[92] Kartik Talamadupula. 2021. Applied AI Matters - AI4Code: Applying Artificial Intelligence to Source Code. *Association for Computing Machinery (ACM) Special Interest Group on AI (SIGAI) AI Matters* 7 (2021). Issue 1.

[93] Michele Tufano, Dawn Drain, Alexey Svyatkovskiy, Shao Kun Deng, and Neel Sundaresan. 2020. Unit Test Case Generation with Transformers. *arXiv preprint arXiv:2009.05617* (2020).

[94] Jennifer Wortman Vaughan and Hanna Wallach. 2020. A human-centered agenda for intelligible machine learning. *Machines We Trust: Getting Along with Artificial Intelligence* (2020).

[95] Jesse Vig. 2019. A Multiscale Visualization of Attention in the Transformer Model. In *ACL*.

[96] Jesse Vig and Yonatan Belinkov. 2019. Analyzing the Structure of Attention in a Transformer Language Model. In *BlackboxNLP@ACL*.

[97] Donald Martin Vinodkumar Prabhakaran Jr. 2020. Participatory Machine Learning Using Community-Based System Dynamics. *Health and Human Rights* 22, 2 (2020), 71.

[98] Abhishek Wadhwani and Priyank Jain. 2020. Machine Learning Model Cards Transparency Review: Using model card toolkit. In *2020 IEEE Pune Section International Conference (PuneCon)*. IEEE, 133–137.

[99] Yue Wang, Weishi Wang, Shafiq R. Joty, and S. Hoi. 2021. CodeT5: Identifier-aware Unified Pre-trained Encoder-Decoder Models for Code Understanding and Generation.

[100] Justin D Weisz, Michael Muller, Stephanie Houde, John Richards, Steven I Ross, Fernando Martinez, Mayank Agarwal, and Kartik Talamadupula. 2021. Perfection Not Required? Human-AI Partnerships in Code Translation. In *26th International Conference on Intelligent User Interfaces*. 402–412.

[101] Sarah Wiegreffe and Yuval Pinter. 2019. Attention is not not explanation. *arXiv preprint arXiv:1908.04626* (2019).

[102] Christine T Wolf. 2019. Explainability scenarios: towards scenario-based XAI design. In *Proceedings of the 24th International Conference on Intelligent User Interfaces*. 252–257.

[103] Enhao Zhang and Nikola Banovic. 2021. Method for Exploring Generative Adversarial Networks (GANs) via Automatically Generated Image Galleries. In *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems*. 1–15.

[104] Bolei Zhou, Aditya Khosla, Àgata Lapedriza, Aude Oliva, and Antonio Torralba. 2016. Learning Deep Features for Discriminative Localization. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2016), 2921–2929.

[105] Haiyi Zhu, Bowen Yu, Aaron Halfaker, and Loren Terveen. 2018. Value-sensitive algorithm design: Method, case study, and lessons. *Proceedings of the ACM on Human-Computer Interaction* 2, CSCW (2018), 1–23.
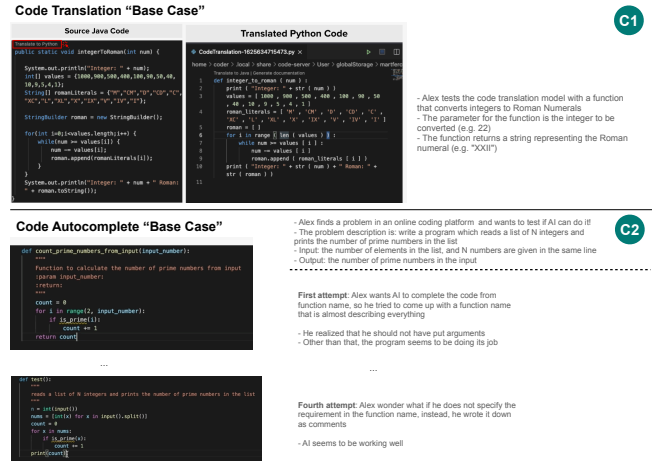
Figure 3: The two other code "base cases" we used in the workshop on Mural. C1 is the example we used as the base case for the workshops about code translation (W1-CT, W2-CT and W5-CT), and C2 is for the code autocompletion use case (W3-CA, W4-CA and W9-CA). The C in Figure 1 is the base code base for the natural language to code use case (W6-NL2Code, W7-NL2Code, W8-NL2Code).

A THE OTHER TWO USE CASES

We show the example code as the base case for workshops about natural language to code (W6-NL2Code, W7-NL2Code, W8-NL2Code) in Figure 1. As a supplement, we show how we replace Figure 1 (c) with other code examples in Figure 3 for the code translation and code autocompletion use cases. The other sub-figures in Figure 1 stay almost the same for all workshops across all use cases with minimum edits.