Identifying & Interactively Refining Ambiguous User Goals for Data Visualization Code Generation

Anonymous ACL submission

Abstract

Establishing shared goals is a fundamental step in human-AI communication. However, ambiguities can lead to outputs that seem correct but fail to reflect the speaker's intent. In this paper, we explore this issue with a focus on the data visualization domain, where ambiguities in natural language impact the generation of code that visualizes data. The availability of multiple views on the contextual (*e.g.* the intended plot and the code rendering the plot) allows for a unique and comprehensive analysis of diverse ambiguity types. We develop a taxonomy of types of ambiguity that arise in this task and propose metrics to quantify them. Using Matplotlib problems from the DS-1000 dataset, we demonstrate that our ambiguity metrics better correlate with human annotations than uncertainty baselines. Our work also explores how multi-turn dialogue can reduce ambiguity, and therefore, improve code accuracy by better matching user goals. We evaluate three pragmatic models to inform our dialogue strategies: Gricean Cooperativity, Discourse Representation Theory, and Questions under Discussion. A simulated user study reveals how pragmatic dialogues reduce ambiguity and enhance code accuracy, highlighting the value of multi-turn exchanges in code generation.

1 Introduction

In human-human interactions, ambiguity resolution has been explored through various well-established frameworks in linguistic pragmatics, such as Rational Speech Act (RSA)(Frank and Goodman, 2012), Discourse Representation Theory (DRT) (Lascarides and Asher, 2007; Kamp et al., 2010), and Questions Under Discussion (QUD) (Roberts, 2012). Further, as Clark (1996) argues, successful interactions depend on establishing common ground by iterative alignment of assumptions and resolving ambiguities on the way to common ground. On the other hand, human-AI collaboration still often fails to provide a human-like pair-programming experience (Williams, 2001; Sarkar et al., 2022), and we claim that a key challenge here is ambiguity resolution—as users' natural language descriptions of intent often map to multiple valid code implementations, requiring iterative clarifications. In this paper, we propose identifying (§3) and then resolving ambiguities (§4) in a dyadic setting of human-LLM interactions using pragmatics-inspired, persona-based prompting.

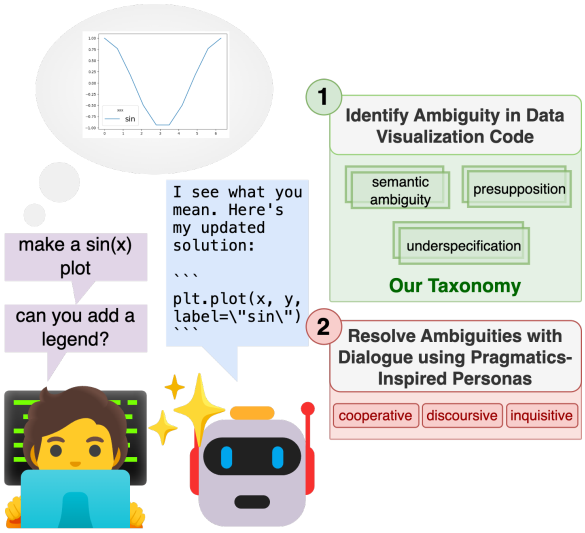

sin
make a sin(x) plot
can you add a legend?
I see what you mean. Here's my updated solution:
```
plt.plot(x, y, label=\"sin\")
```
1
Identify Ambiguity in Data Visualization Code
semantic ambiguity
presupposition
underspecification
Our Taxonomy
2
Resolve Ambiguities with Dialogue using Pragmatics-Inspired Personas
cooperative
discoursive
inquisitive

Figure 1: This figure summarizes the contributions of this paper. We formalize and identify ambiguity in data visualization code, then use pragmatics-inspired dialogue strategies to interactively resolve ambiguities in user intents. To this end, we present a multimodal taxonomy of ambiguity categories, and new metrics informed by this taxonomy.

To achieve this goal, we frame the natural language to code problem as a two-player cooperative dialogue. A **director** (typically the user) specifies their intent in natural language and a **coder** (typically an automated coding assistant) generates code with the functionality the director had in mind. A goal for this pair-programming setting is to have a coder agent that can interact with the director agent to resolve ambiguity and generate code. While, in principle, the coder's uncertainty in this task can come from many sources, focus is typically placed on the inherent model limitations of the coder agent caused by insufficient knowledge or training. In contrast, this work focuses on the uncertainty of the coder about the user's goals—*i.e.*, the *ambiguity* of director's requests. We focus on how the director can resolve the coder's uncertainty through clear communication about intended goals, and how coders can clarify goals through conversation.

Based on our definition of ambiguity, we hypothesize that ambiguity reduction should lead to improved code accuracy. To study this, we propose a taxonomy that more precisely codifies what is meant by ambiguity. Our taxonomy includes categories of ambiguity, and examples specific to the plotting domain — where the presence of multimodal contexts allows us to explore more diverse sources of natural language ambiguity (*e.g.*. We propose a number of metrics based on our taxonomy that allow us to dynamically (and automatically) quantify natural language ambiguity in text-to-code problems. We use our definition of ambiguity to study how multi-step communication with the director can serve to reduce ambiguity. Guided by theories of pragmatics we simulate dialogue between two machine agents. This shows how (pragmatic) dialogues can help coders resolve ambiguity, and ultimately, improve task success.

The contributions of this paper are as follows:

1. We propose a taxonomy to codify ambiguity in multimodal text-to-code problems;
2. We propose a number of potential metrics to measure these defined notions of ambiguity, and empirically test which best represents different aspects of ambiguity;
3. We propose strategies, inspired by theories of discourse, for incorporating dialogue in a coding agent to reduce ambiguity and increase task success.

From our analyses, we find that our metrics can predict ambiguity categories, and pragmatic dialogue increases task success (measured by correctness of the generated code) while targeting the identified ambiguities. We make our code, and annotations publicly available for the camera-ready version of this paper.

2 Related Work

Code Generation Large language models of code have shown strong performance on natural language to code generation (Chen et al., 2021; Rozière et al., 2024; Lozhkov et al., 2024, *inter alia*). However, work investigating how users interact with code generation models has found that impressive benchmark performance does not always translate to improved task outcomes for users (Sarkar et al., 2022; Vaithilingam et al., 2022; Ma et al., 2023; Mozannar et al., 2024; Nguyen et al., 2024). Some of this gap can be attributed to the ambiguity inherent to human interactions with code models: Sarkar et al. (2022) observe that user utterances are often underspecified and ambiguous, forcing users to repeatedly refine their prompts and adapt their thought processes to match the LLM. Likewise, Mozannar et al. (2024) observe that users often provide fuzzy instructions, motivating a clarification feature. Underspecified instructions are also present in real-world software engineering benchmarks such as SWE-bench (Jimenez et al., 2024), as found during the construction of SWE-bench Verified.[1]

Recent work has studied ambiguity resolution for code LLMs via clarification questions. Mu et al. (2024) introduce ClarifyGPT, a pipeline for code generation with selective clarification. Li et al. (2023) studies clarification for open-domain code generation in a scaffolded setting. Also, uncertainty estimation and explainability literature have been instrumental for quantifying ambiguity in the works by Liu et al. (2024) and Lin et al. (2024). Further, explorations of ambiguity in SQL generation (Bhaskar et al., 2023; Wang et al., 2023) and the role of ambiguity in requirements engineering (Kamsties, 2005; Bano) have been recently studied.

Ambiguity in NLP Tasks Ambiguity has been studied across a wide array of NLP tasks, including coreference resolution (Poesio and Artstein, 2005), (visual) question answering (Min et al., 2020; Kim et al., 2023; Papakostas and Papadopoulou, 2023; Park et al., 2024) and machine translation (Iyer et al., 2023; Schouten et al., 2023; Niwa and Iso, 2024; Madureira et al., 2024). Current language models generally struggle when applied directly to tasks with ambiguity (Liu et al., 2023; Zhang et al., 2024; Wildenburg et al., 2024); by default, they do

---

[1] `https://openai.com/index/introducing-swe-bench-verified/`

not recognize ambiguity in instructions, nor do they seek clarification or engage in proactive dialogue to resolve ambiguity (Deng et al., 2023). However, recent sampling-based methods have shown promise in detecting ambiguity (Kadavath et al., 2022; Kuhn et al., 2023b; Cole et al., 2023; Lin et al., 2024), while prompting and self-improvement methods have proven effective for clarifying ambiguity with LLMs (Krasheninnikov et al., 2022; Kuhn et al., 2023a; Andukuri et al., 2024).

Pragmatics One approach to resolving ambiguity is to assume the speaker is a rational agent playing a cooperative game (Grice, 1975) where they are choosing an utterance that gives the code generation model the best chance of recovering the program they have in mind. This form of inference has been formalized in the Rational Speech Acts (RSA) framework (Frank and Goodman, 2012). RSA has been productively applied to programming tasks where a user specifies their intent using examples (Pu et al., 2020, 2023; Vaduguru et al., 2024). Similar approaches to disambiguation also been applied to code generation from natural language using large language models (Zhang et al., 2023). Other pragmatic theories of discourse work include RSA for referential communication in a game of color (Monroe et al., 2017; McMahan and Stone, 2020), question under discussion (Ko et al., 2023), and discourse theories as applied to dialogue settings (Asher et al., 2016; Chi and Rudnicky, 2022; Atwell et al., 2021, 2024, 2022). The frameworks we use to implement our dialogue agents are inspired by these in this work.

3 Defining and Identifying Ambiguity

We define ambiguity to arrive at a taxonomy that helps us identify it automatically.

Director-Coder Setting Initially, we formalize the setting described in § 1. The **director** has a target intent I – a random variable representing the goal image (or corresponding code)[2] – which the director communicates through a natural language utterance $U(I)$. The utterance U is also a random variable. The **coder** interprets this utterance to directly generate code (and corresponding image) $\hat{I} \sim \mathbb{C}(U)$, where $\mathbb{C}(U)$ is a code distribution conditional to the utterance U. We assume there is some evaluation method that produces a random error $E(I, \mathbb{C}(U))$ to quantify goal fulfillment in this context. For example, in later experiments, we use k repeated samples $\hat{I} \sim \mathbb{C}(U)$ and check the accuracy of the code by counting how many pass unit tests (pass@k). In this definition, we will assume that the minimization of the error, E, is preferable.

3.1 Ambiguity in Plotting Code

Definition We define ambiguity from the perspective of the coder. Intuitively, we frame ambiguity as any portion of uncertainty held by the coder that could be reduced by changing the natural language utterance U. More specifically, interpreting E as a form of error, we define ambiguity as the quantity:

$$\begin{aligned} & E(I, \mathbb{C}(U)) - E(I, \mathbb{C}(U^*)) \\ & \text{where} \quad U^* = \arg\min_{U' \in \Upsilon} \ E(I, \mathbb{C}(U')), \end{aligned} \tag{1}$$

and Υ is a constraining set to ensure director utterances are "reasonable." For instance, Υ could be the set of all finite, grammatically correct utterances in the English language.

Interpretation In a typical machine learning problem, a model h is picked from some constraining set model class $\mathcal{H}$, selecting this model to reduce error as much as possible. For instance, we can select a linear model h from the set $\mathcal{H}$ of all linear models parameterized by elements of $\mathbb{R}^d$. Meanwhile, there is also a best model h^* that minimizes the error for our problem.[3] This minimum error (or uncertainty) is simply irreducible without changing the problem definition. Here, we adapt this to define ambiguity. The term $E(I, \mathbb{C}(U^*))$ represents the minimum uncertainty, treating the utterance itself U as the model h we wish to pick, within the aforementioned problem. In turn, ambiguity is formally defined *as the excess error (or uncertainty) of the coder that could have been reduced through improved communication by the director*. This quantity formalizes the process of disambiguation – studied throughout this paper – and what it means to disambiguate for a coder with fixed knowledge.

Taxonomy Next, we use this definition to arrive at a taxonomy of ambiguities in the plotting domain. We observe that different input modalities to LLMs inherently convey different types of ambiguity (see Figure 2), as these modalities serve different user

---

[2] We do not differentiate between these, since each code corresponds to an image.

[3] The best model h^* is called the *Bayes optimal* model and the error of this model is the *Bayes error* or the *aleatoric uncertainty* (Hüllermeier and Waegeman, 2021).

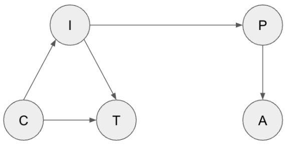


Figure 2: This figure depicts the causal graph of a multi-modal coding task. I: Image, C: ground truth code used to generate I, T: unit tests, P: director's prompt, and A: coder's answer. When ambiguity is prevalent, then the edges of this graph are cut, and the coder's answer relies on a non-descriptive prompt.

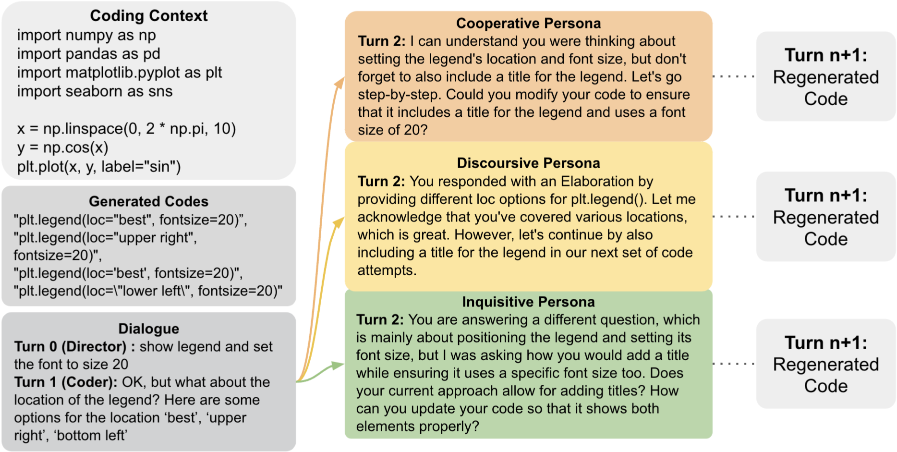
intents. For instance, an image in the "mind's eye" of the director corresponds to an ideal end product, but the director may not know how this translates to precise code or the best ways to communicate their ideal. This type of cross-modality dependency has been studied by the discourse coherence literature (Alikhani et al., 2020; Inan and Alikhani, 2024). In order to represent these cross-modal dependencies, we need a taxonomy that captures the different types and sources of ambiguity, which we present next. These categories draw inspiration from cognitive science, linguistics, and discourse literatures, as well as our definition.

- **Semantic ambiguity:** Certain words and their meanings can have multiple interpretations. Thus, misinterpretation of U by the coder may lead to errors that would be prevented with a better choice of words (i.e., the optimal utterance U^*), implying higher ambiguity (Eq. 1). This category is based on semantic ambiguity in cognitive science literature as explored by Zempleni et al. (2007); Degani and Tokowicz (2010).
  e.g. "regular matplotlib style plot", "grouped histogram", "color plot"
- **Presupposition:** Instructions may presuppose default parameter values without explicitly mentioning their use, and disagreement in coder and director presuppositions can also be a source of error caused by the coder's interpretation of U, leading to increased ambiguity. This category is based on the presuppositional ambiguity as explored by Zuber (1973); Atlas (1977); Kempson (1979); Jaszczolt (2002).
  e.g. knowing the default parameters of the `scatter()` *function, or original instructions ask to "keep the distribution plot in blue", but no explicit parameter for "blue" is made because the default is known to be blue.*
- **Underspecification:** Some non-default parameter values or functions are either not mentioned at all, or partially described in the instructions. By leaving out these details, a coder's own interpretation of U can be counter to the director's goals, once again increasing the error compared to U^* and the ambiguity as defined previously. This category is based on the relationship between underspecification and ambiguity as explored by van Deemter and Peters (1996). Some context-specific subcategories for scientific plotting can include,
  - **Color parameter**, *e.g. "solid red", alpha value is set but not mentioned*
  - **Distance parameter**, *e.g. "enough" space between axes*
  - **Shape parameter**, *e.g. marker shape is set even though not mentioned*
  - **Size parameter**, *e.g. marker size is set even though not mentioned*
  - **Location parameter**, *e.g. legend location is set even though not mentioned*
  - **Label parameter**, *e.g. title is set to be "xxx" even though not mentioned*
  - **Line parameter**, *e.g. "full line", "dashed line"*
  - **Function**, *e.g. to plot a heatmap, using either* `imshow()` *or* `pcolor()`.

Annotating for Ambiguity We carry out an annotation and do a preliminary analysis of the distribution of these categories of ambiguity in the DS1000 dataset (Lai et al., 2022), specifically with questions from plotting libraries (e.g., `Matplotlib`, `Seaborn`) (Hunter, 2007). This dataset, featuring natural language prompts from StackOverflow with human-written tests and plots, inherently contains potential human-introduced ambiguities. We, the authors, who have prior experience with ambiguity in dialogue, annotated 155 coding instructions from the DS1000 dataset. Despite the DS1000 dataset's claim that the problems are written to be unambiguous, we still find that 57% of the plotting questions fall under one or more of the categories we have defined above. Interannotator agreement on 25 sampled questions, measured by Cohen's Kappa ($\kappa = 0.587$), indicates moderate to substantial agreement, as anticipated for this subjective meta-labeling task (Fleiss and Cohen, 1973; Banerjee et al., 1999; Viera et al., 2005). Table 1 shows the distribution of different categories.

3.2 Automatic Metrics for Ambiguity

In addition to the human annotations, we propose automatic ways of measuring ambiguity based on our formalizations and taxonomy (§3.1), and compare them to traditionally employed uncertainty-based metrics.



Coding Context

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

x = np.linspace(0, 2 * np.pi, 10)
y = np.cos(x)
plt.plot(x, y, label="sin")
```

Generated Codes

```
"plt.legend(loc="best", fontsize=20)",
"plt.legend(loc="upper right", fontsize=20)",
"plt.legend(loc='best', fontsize=20)",
"plt.legend(loc=\"lower left\", fontsize=20)"
```

Dialogue

Turn 0 (Director) : show legend and set the font to size 20

Turn 1 (Coder): OK, but what about the location of the legend? Here are some options for the location ‘best’, ‘upper right’, ‘bottom left’

Cooperative Persona

Turn 2: I can understand you were thinking about setting the legend's location and font size, but don't forget to also include a title for the legend. Let's go step-by-step. Could you modify your code to ensure that it includes a title for the legend and uses a font size of 20?

Discoursive Persona

Turn 2: You responded with an Elaboration by providing different loc options for plt.legend(). Let me acknowledge that you've covered various locations, which is great. However, let's continue by also including a title for the legend in our next set of code attempts.

Inquisitive Persona

Turn 2: You are answering a different question, which is mainly about positioning the legend and setting its font size, but I was asking how you would add a title while ensuring it uses a specific font size too. Does your current approach allow for adding titles? How can you update your code so that it shows both elements properly?

Figure 3: This figure shows the dialogue flow for a pragmatic director, where the initial intent of the dialogue is given on the left, and the different responses generated using separate personas are given in the middle.

Ambiguity Category	Distribution
semantic ambiguity	23.8%
presupposition	11.9%
underspecification	73.8%

Table 1: This table shows the ambiguity category distribution within the ambiguous prompts (57%) of the 155 plotting questions in the DS1000 dataset based on our annotations.

Sampling Diversity (SD) A common approach (Cole et al., 2023) to measuring uncertainty about a given utterance U is to count the number of distinct programs that satisfy the constraints specified in the prompt. If the coder's solution distribution $\mathbb{C}(U)$ assigns non-zero probability to many codes (i.e., it generates many different codes upon sampling), then these codes all differ in apsects unconstrained by the prompt U from the coder's perspective. This indicates the coder is uncertain about the intent of U. The more distinct programs there are for a given sample size, the higher the coder's uncertainty about the utterance is. This method serves as a baseline for ambiguity measurement; while it does capture uncertainty about the utterance U, it does not capture information about the optimal prompt U^* in any way. This is an important distinction between typical notions of uncertainty and our proposed definition of ambiguity. We give the implementation details in Appendix §D, where we compare Abstract Syntax Trees (AST) of both programs using edit distance.

Repeated Parameter Counting (RPC) Another way to quantify ambiguity in an utterance U is to focus on identifying function calls/parameters that appear across the set of proposed programs. This is conceptually similar to sampling diversity, which compares distinct solutions directly. We hypothesize function calls and parameters may better capture presuppositional ambiguity within our taxonomy since this ambiguity is directly related to the default parameter settings within a code library. More specifically, our proposed RPC metric measures ambiguity by counting which function calls and parameters remain constant across the code solutions. The fewer elements that must remain fixed, the higher the ambiguity. Similar to sample diversity, this metric does not explicitly consider the optimal prompt U^*. We expect it to perform well for presupposition, because it focuses on aspects of code that are often presupposed.

Optimal Result Gap (ORG) Building on our definition in Eq. 1, we propose a subfamily of metrics that attempt to directly compute this mathematical notion of ambiguity. While the coder's realized uncertainty $E(I, \mathbb{C}(U))$ is easy to compute by using Pass@k scores for a given utterance U, the minimum uncertainty $E(I, \mathbb{C}(U^*))$ is more difficult because we cannot be sure of the optimal utterance U^*. We propose to approximate U^* by using one of three oracle information sources: ground-truth code for the plot, ground-truth image for the plot, and unit tests for the "Pass" determination. We provide these oracle sources to a large language model and ask it to provide a prompt that enable itself to generate the code, create the image with code, or generate code that would pass the tests. While this approach provides an estimate, we acknowledge its limitations: the model-generated re-prompt may not fully capture an ideal, ambiguity-free prompt, making it an imperfect but practical approximation of the optimal formulation. At the same time, we expect it to improve over the previous two metrics because it does consider the optimal prompt U^*.

Self Verification This is a traditional method of uncertainty estimation where a model is asked to return its uncertainty about a solution when provided a problem statement. This baseline uncertainty methodology, like sampling diversity and RPC, does not explicitly consider the optimal utterance U^*, which is an important aspect of our definition. We use a prompt similar to the ones used by Cole et al. (2023) and Sicilia et al. (2024).

LLM-Based Ambiguity Rating (LAR) On the other hand, we can extend self-verification to better consider our definitions of ambiguity. Instead of querying the model for its uncertainty, we can prompt it to rate the ambiguity of U on a scale of 1 to 10, providing the model with our ambiguity taxonomy as a resource in the prompt. This encourages the rating aligning with our pre-defined ambiguity categories, rather than being an arbitrary self-assessment. Since these categories are also based on our initial mathematical definition of ambiguity, it also serves as heuristic approximation of Eq. 1. Different from baseline methods focused on uncertainty, it implicitly considers the optimal prompt U^* through the characterizations of optimality encoded in our taxonomy.

4 Disambiguation with Coding Dialogues

After formalizing ways of identifying ambiguity, we now propose that ambiguity can be resolved using dialogues, and we formalize a dialogue setup with persona-based generation components.

4.1 Basic Dialogue Setup

We propose resolving ambiguity in natural language specifications of intent with multi-turn dialogue. Each coding task is defined by the natural language intent I (see Figure 3) and the initial request U_1, as before. Meanwhile, our proposal extends the previous setting to allow for dialogue:

1. Director presents instruction $U_1(I)$.
2. Coder responds with an utterance $U_2(U_1)$.
3. Director continues $U_i(U_{:i-1}, I)$, using access to target image and utterance history.
4. Coder continues $U_{i+1}(U_{:i-1})$, using access to utterance history only.
5. Repeat steps 3 and 4 based on the number of turns controlled as a hyperparameter.
6. Session always terminates with coder providing their final code $\hat{I}(U_{:})$.

In this setting, coder utterances can include dialogue acts like clarification questions, e.g., “C: what location should I put the legend,” which evokes a specific response, “D: The top left corner” or can be more general declarations that start a sub-topic of conversation “C: I’ll plan for the default legend arguments. D: Hmm. Keep it on the top left. What else can you change?”

4.2 A Pragmatic Dialogue Setup

We hypothesize ambiguity reduction in our proposed dialogue setting can be improved by encouraging LLM-based coders to consider pragmatics in their dialogue strategy. We operationalize this by using persona prompting and in-context learning as described in Wang et al. (2024); Schulhoff et al. (2024), and Zheng et al. (2024). Next, we describe the personas we use to generate responses. These are based on three theories of discourse: cooperative, discoursive, and inquisitive. For the implementation details of this setup, please refer to Appendix A.

4.2.1 Dialogue Strategies

Cooperative The first framework we use is based on Grice’s maxims of cooperative dialogue partners (Grice, 1975; Horn, 1984; Levinson, 2000; McMahan and Stone, 2020). Here, the coder is a pragmatic agent that recursively engages in interaction and models the director’s state of mind to respond to an utterance. We use the Gricean cooperativity principle to design the prompt for this dialogue strategy, given in Appendix B.

Discoursive The second pragmatics framework is based on Discourse theories. Here, the coder is not necessarily responding strategically. Still, it’s utterance is always related to the history of the conversation and the coding context through a set of coherence relations. Hence, when a coder produces an utterance, it relates to the set of solutions it has sampled as well as what the director has said in the previous turn. This definition of discourse is mostly similar to SDRT-like dialogue-based relation categories (Ko et al., 2023; Asher et al., 2016; Fu, 2022; Atwell et al., 2024; Alikhani et al., 2023). We provide the persona prompt in Appendix B.

Inquisitive The third pragmatics framework is related to discourse theories, but focuses more on question-type relations. In this case, the coders

	Sem. Amb.	Underspec.	Presup.	Avg.
RPC	0.450	0.412	0.466	0.443
ORG$_C$	0.527	0.495	0.326	0.449
ORG$_I$	0.597	0.450	0.451	0.499
ORG$_U$	0.561	0.494	0.445	0.500
LAR	0.655	0.453	0.447	0.518
LAR$_\mathcal{T}$	0.585	0.530	0.622	0.579
SV	0.380	0.399	0.493	0.424

Table 2: This table shows the AUC scores between different ambiguity metrics we propose and the ambiguity categories from our taxonomy. This shows that most metrics are predictive of semantic ambiguity, while underspecification and presupposition are less correlated. Subscripts indicate the ground truth modality: I (images), C (code), U (unit test).

utterance explicitly answers an implicit question posed by the director. This discourse framing is described by Clifton and Frazier (2012) under the umbrella term of Questions Under Discussion (QUD). When a director gives a coding instruction, the pragmatic coder with QUD understanding first detects an implied question indirectly posed by the director and the coder answers that question. The persona prompt is given in Appendix B.

5 Experiments & Findings

In this section, we provide details of our experiments of disambiguation, and user simulation and show the utility of our ambiguity taxonomy and metrics. We answer multiple research questions and report our findings in combination with our experiments. We first describe the experiments for our taxonomy and automatic metrics as described in Section §3.1, and then follow up with experimentation based on the dialogue approach to coding we described in Section §4. We experiment mainly with GPT-4o in our experiments, but provide additional results for LLaMA-3.2, StarCoder, and CodeLLaMA in Appendix §C.

Our Metrics Predict Ambiguity Categories To test the hypothesis of whether our metrics are predictive of ambiguity (as we defined), we carry out a correlation study using ROC AUC scores[4] (Table 2). In our case, we use it to measure correlation between the ambiguity scores and the ground-truth human-produced ambiguity labels for each instance of the 155 plotting questions. We observe

		Pass@1
Baseline (no dialogue)		68.38%
With Reference Code	Cooperative	79.44%
	Discoursive	74.11%
	Inquisitive	66.34%
With Reference Image	Cooperative	75.23%
	Discoursive	74.06%
	Inquisitive	64.56%
Ceiling Performance (Non-Ambiguous Reprompt)		87.74%

Table 3: This table presents the mean pass@1 scores for different types of dialogue strategies that we propose (§4). The baseline corresponds to the GPT-4o code answers to the original prompts, while the ceiling performance uses non-ambiguous reprompts. Having a dialogue with pragmatics-inspired personas improves task success drastically, yet there is still ambiguity between the ceiling performance.

that ORG, which contains the oracle solutions to approximate ambiguity, predicts the semantic ambiguity category, but less so the other categories. This may be due to the class imbalance in the dataset, and also the suboptimal nature of reprompts used in the ORG metrics. The most predictive of any ambiguity category is the LAR$_\mathcal{T}$ metric, where a model is asked to rate the ambiguity using our taxonomy. This shows the validity and applicability of this metric to unsupervised contexts. The lowest prediction power comes from the traditional uncertainty measurement technique of self-verification (SV), as it does not necessarily correlate with the ambiguity of the user's intent, but the uncertainty of the model providing an answer to the prompt.

Pragmatic Dialogue Increases Task Success To test the hypothesis of whether the pragmatic dialogue setup that we proposed in §4 disambiguates and improves task success (as measured by the pass@k correctness score), we carry out a comparative experiment with results shown in Table 3. Here, we test two scenarios, one in which the director is given the reference code and one where the reference image is used. This comparison alleviates the concern about whether there is ground truth code leakage from the director to the coder. It can be observed that the best-performing dialogue strategy is pragmatic cooperative reasoning in both categories, likely benefiting from the theory-of-mind reasoning and chain-of-thought training in modern LLMs.

The inquisitive strategy is the least-performing

---

[4] Area Under the ROC Curve (**AUC**) is a measure of correlation about how well a continuous independent variable can predict a binary dependent variable, via testing a variety of different thresholds. An AUC of 0.5 is equivalent to a random baseline, while an optimal score is 1.

Coding Question	Ambiguity	Baseline	Cooperative	Discoursive	Inquisitive
draw a line (with random y) for each different line style	underspecification	0.000	0.000	0.200	0.000
draw a full line from (0,0) to (1,2)	semantic ambiguity	0.000	0.067	0.000	0.000
make seaborn relation plot and color by the gender field of the dataframe df	underspecification	0.067	0.533	0.000	0.000
highlight in red the x range 2 to 4	semantic ambiguity	0.667	0.967	1.000	0.167

Table 4: This table shows a breakdown of the final executability scores (pass@1 with 30 samples each instance) for different questions in the DS1000 dataset, with their annotated ambiguity categories. The examples are picked to show when most models have low scores, or to show the performance according to different categories of ambiguity.

model, even worse than the baseline, which may mean that always looking for questions under the discussion can hinder disambiguation. Despite improvements, a gap remains between the best-performing strategy and the ceiling performance, indicating unresolved ambiguity. Since user intent is fixed in this static task, full resolution is unlikely, and even the ceiling performance is imperfect, as the re-prompt itself may still contain ambiguity.

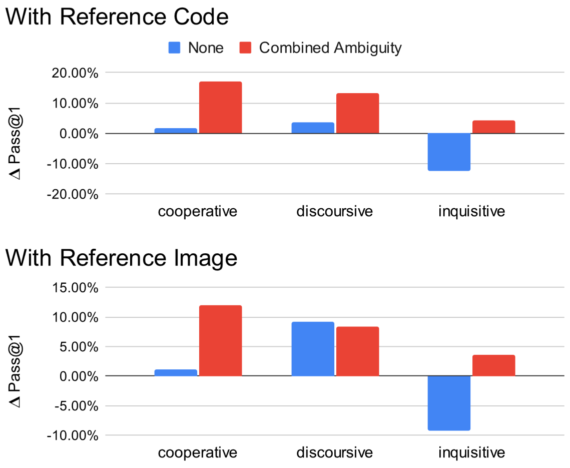


Figure 4: This figure shows a breakdown of the change in the mean pass@1 scores (Δ = post-dialogue − original) across ambiguous and non-ambiguous instances of the DS1000 dataset. Dialogue shows better performance in ambiguous instances instead of non-ambiguous ones.

Pragmatic Dialogue Targets Ambiguities To evaluate whether dialogue strategies improve code accuracy by directly addressing the ambiguities identified in our taxonomy, we measure the change in mean pass@1 scores between ambiguous and non-ambiguous cases (Figure 4)[5]. The results clearly show that dialogue-driven improvements are consistently greater for ambiguous cases than for non-ambiguous ones. This confirms that dialogue effectively disambiguates prompts across all three pragmatic personas. However, when the reference image is provided instead of the code, the Discoursive Persona performs similarly in both cases, suggesting that dialogue alone may not fully resolve ambiguities. Additionally, in line with Table 3, Δpass@1 for the Inquisitive Persona is negative in non-ambiguous instances, yet it still succeeds in clarifying ambiguous prompts.

5.1 Error Analysis

Table 4 presents a detailed performance breakdown of different dialogue strategies. Notably, certain questions remain challenging even after dialogue, yet specific ambiguity categories align with the most effective pragmatic strategy. For instance, nearly all personas failed to resolve the first underspecification question (mean pass@1: 0.000), with only the cooperative persona achieving occasional success (mean pass@1: 0.267). Interestingly, in some cases, additional dialogue negatively impacted performance across all personas. The inquisitive persona performed best for vagueness-related ambiguities, while the discoursive and cooperative personas excelled in addressing parameter underspecification.

6 Conclusion

Overall, in this paper, we have proposed a dialogue-oriented perspective to code generation. We characterized various pragmatics frameworks in relation to pair-programming-like dialogues that happen between a director and a coder. We then analyzed the effects of having dialogues with different reasoning strategies on the executability and disambiguation of the final generated code. As having a dialogue based on code is becoming the norm with LLMs, focusing on the pragmatics of dialogue opens up new venues for developing dialogue systems, datasets, and evaluation mechanisms for code generation.

---

[5] We focus specifically on task success rather than user satisfaction due to the subjectivity and costs of user experiments, while already showing that ambiguity is addressed.

Limitations

We proposed using pragmatic dialogue for code generation, but the major limitation is from the side of human data collection and evaluation. We resorted to automatic metrics already being used or developed for this study to evaluate our setup without relying on human annotators. However, this entails that the evaluations may not be human-like and may not show the most accurate representations even though they show improvements in generally accepted code executability standards. Further, we did not deploy a dialogue system to study our approach. Instead, we resorted to simulations using LLMs, which may or may not accurately represent how a human interlocutor would act in a real-world setting. We wanted to minimize this by using large parameter models for dialogue generation and StackOverflow-based code instructions from the DS1000 dataset.

Ethics Statement

In our simulation process we have used GPT-4o, and this is a closed-source LLM, and we are aware that this model can propagate its own training biases. The scientific community does not have access to any information regarding how this model is trained or what the dataset consists of. This may result in a deficient evaluation of the final performance and human-likeness of the generated dialogue. This is a simulated analysis study to identify and characterize pragmatics frameworks with possible LLM behavior in a pair programming setting. Hence, we do not involve humans in our current setup. The biases propagated by GPT-4o are the responsibility of OpenAI and should be held accountable by their and the scientific community's ethical standards.

References

Malihe Alikhani, Baber Khalid, and Matthew Stone. 2023. Image–text coherence and its implications for multimodal AI. *Front. Artif. Intell.*, 6:1048874.

Malihe Alikhani, Piyush Sharma, Shengjie Li, Radu Soricut, and Matthew Stone. 2020. Cross-modal coherence modeling for caption generation. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 6525–6535, Online. Association for Computational Linguistics.

Chinmaya Andukuri, Jan-Philipp Fränken, Tobias Gerstenberg, and Noah D. Goodman. 2024. Star-gate: Teaching language models to ask clarifying questions. *ArXiv*, abs/2403.19154.

Nicholas Asher, Julie Hunter, Mathieu Morey, Benamara Farah, and Stergos Afantenos. 2016. Discourse structure and dialogue acts in multiparty dialogue: the STAC corpus. In *Proceedings of the Tenth International Conference on Language Resources and Evaluation (LREC'16)*, pages 2721–2727, Portorož, Slovenia. European Language Resources Association (ELRA).

Jay David Atlas. 1977. Negation, ambiguity, and presupposition. *Linguistics and Philosophy*, 1(3):321–336.

Katherine Atwell, Remi Choi, Junyi Jessy Li, and Malihe Alikhani. 2022. The role of context and uncertainty in shallow discourse parsing. In *Proceedings of the 29th International Conference on Computational Linguistics*, pages 797–811, Gyeongju, Republic of Korea. International Committee on Computational Linguistics.

Katherine Atwell, Mert Inan, Anthony B. Sicilia, and Malihe Alikhani. 2024. Combining discourse coherence with large language models for more inclusive, equitable, and robust task-oriented dialogue. In *Proceedings of the 2024 Joint International Conference on Computational Linguistics, Language Resources and Evaluation (LREC-COLING 2024)*, pages 3538–3552, Torino, Italia. ELRA and ICCL.

Katherine Atwell, Junyi Jessy Li, and Malihe Alikhani. 2021. Where are we in discourse relation recognition? In *Proceedings of the 22nd Annual Meeting of the Special Interest Group on Discourse and Dialogue*, pages 314–325, Singapore and Online. Association for Computational Linguistics.

Mousumi Banerjee, Michelle Capozzoli, Laura McSweeney, and Debajyoti Sinha. 1999. Beyond kappa: A review of interrater agreement measures. *Can. J. Stat.*, 27(1):3–23.

Muneera Bano. Addressing the challenges of requirements ambiguity: A review of empirical literature. In *2015 IEEE Fifth International Workshop on Empirical Requirements Engineering (EmpiRE)*, page 24. IEEE.

Adithya Bhaskar, Tushar Tomar, Ashutosh Sathe, and Sunita Sarawagi. 2023. Benchmarking and improving text-to-SQL generation under ambiguity. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 7053–7074, Singapore. Association for Computational Linguistics.

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. *Preprint*, arXiv:2107.03374.

Ta-Chung Chi and Alexander Rudnicky. 2022. Structured dialogue discourse parsing. In *Proceedings of the 23rd Annual Meeting of the Special Interest Group on Discourse and Dialogue*, pages 325–335, Edinburgh, UK. Association for Computational Linguistics.

Herbert H. Clark. 1996. *Using Language*. Cambridge University Press, Cambridge, England, UK.

Charles Clifton, Jr. and Lyn Frazier. 2012. Discourse Integration Guided by the 'Question under Discussion'. *Cognit. Psychol.*, 65(2):352.

Jeremy Cole, Michael Zhang, Daniel Gillick, Julian Eisenschlos, Bhuwan Dhingra, and Jacob Eisenstein. 2023. Selectively answering ambiguous questions. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 530–543, Singapore. Association for Computational Linguistics.

Tamar Degani and Natasha Tokowicz. 2010. Semantic Ambiguity within and across Languages: An Integrative Review. *Q. J. Exp. Psychol.*, 63(7):1266–1303.

Yang Deng, Lizi Liao, Liang Chen, Hongru Wang, Wenqiang Lei, and Tat-Seng Chua. 2023. Prompting and evaluating large language models for proactive dialogues: Clarification, target-guided, and non-collaboration. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pages 10602–10621, Singapore. Association for Computational Linguistics.

Joseph L. Fleiss and Jacob Cohen. 1973. The Equivalence of Weighted Kappa and the Intraclass Correlation Coefficient as Measures of Reliability. *Educational and Psychological Measurement*, 33(3):613–619.

Michael C. Frank and Noah D. Goodman. 2012. Predicting pragmatic reasoning in language games. *Science*, 336(6084):998–998.

Yingxue Fu. 2022. Towards unification of discourse annotation frameworks. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics: Student Research Workshop*, pages 132–142, Dublin, Ireland. Association for Computational Linguistics.

H. P. Grice. 1975. Logic and Conversation. In *Speech Acts*, pages 41–58. Brill, Leiden, The Netherlands.

Laurence Horn. 1984. Toward a new taxonomy for pragmatic inference: Q-based and r-based implicature. *Meaning, Form, and Use in Context: Linguistic Applications*, page 11–42.

Eyke Hüllermeier and Willem Waegeman. 2021. Aleatoric and epistemic uncertainty in machine learning: An introduction to concepts and methods. *Machine learning*, 110(3):457–506.

J. D. Hunter. 2007. Matplotlib: A 2d graphics environment. *Computing in Science & Engineering*, 9(3):90–95.

Mert Inan and Malihe Alikhani. 2024. Seeing eye-to-eye: Cross-modal coherence relations inform eye-gaze patterns during comprehension & production. In *Proceedings of the 2024 Joint International Conference on Computational Linguistics, Language Resources and Evaluation (LREC-COLING 2024)*, pages 14494–14512, Torino, Italia. ELRA and ICCL.

Vivek Iyer, Pinzhen Chen, and Alexandra Birch. 2023. Towards effective disambiguation for machine translation with large language models. In *Proceedings of the Eighth Conference on Machine Translation*, pages 482–495, Singapore. Association for Computational Linguistics.

K. M. Jaszczolt. 2002. Against ambiguity and underspecification: evidence from presupposition as anaphora. *Journal of Pragmatics*, 34(7):829–849.

Carlos E Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik R Narasimhan. 2024. SWE-bench: Can language models resolve real-world github issues? In *The Twelfth International Conference on Learning Representations*.

Saurav Kadavath, Tom Conerly, Amanda Askell, Tom Henighan, Dawn Drain, Ethan Perez, Nicholas Schiefer, Zac Hatfield-Dodds, Nova DasSarma, Eli Tran-Johnson, Scott Johnston, Sheer El-Showk, Andy Jones, Nelson Elhage, Tristan Hume, Anna Chen, Yuntao Bai, Sam Bowman, Stanislav Fort, Deep Ganguli, Danny Hernandez, Josh Jacobson, Jackson Kernion, Shauna Kravec, Liane Lovitt, Kamal Ndousse, Catherine Olsson, Sam Ringer, Dario Amodei, Tom Brown, Jack Clark, Nicholas Joseph, Ben Mann, Sam McCandlish, Chris Olah, and Jared Kaplan. 2022. Language models (mostly) know what they know. *Preprint*, arXiv:2207.05221.

Hans Kamp, Josef Van Genabith, and Uwe Reyle. 2010. Discourse representation theory. In *Handbook of Philosophical Logic: Volume 15*, pages 125–394. Springer.

Erik Kamsties. 2005. Understanding Ambiguity in Requirements Engineering. In *Engineering and Managing Software Requirements*, pages 245–266. Springer, Berlin, Germany.

Ruth M. Kempson. 1979. Presupposition, Opacity, and Ambiguity. In *Presupposition*, pages 283–297. Brill, Leiden, The Netherlands.

Gangwoo Kim, Sungdong Kim, Byeongguk Jeon, Joonsuk Park, and Jaewoo Kang. 2023. Tree of clarifications: Answering ambiguous questions with retrieval-augmented large language models. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 996–1009, Singapore. Association for Computational Linguistics.

Wei-Jen Ko, Yating Wu, Cutter Dalton, Dananjay Srinivas, Greg Durrett, and Junyi Jessy Li. 2023. Discourse analysis via questions and answers: Parsing dependency structures of questions under discussion. In *Findings of the Association for Computational Linguistics: ACL 2023*, pages 11181–11195, Toronto, Canada. Association for Computational Linguistics.

Dmitrii Krasheninnikov, Egor Krasheninnikov, and David Krueger. 2022. Assistance with large language models. In *NeurIPS ML Safety Workshop*.

Lorenz Kuhn, Yarin Gal, and Sebastian Farquhar. 2023a. Clam: Selective clarification for ambiguous questions with generative language models. *Preprint*, arXiv:2212.07769.

Lorenz Kuhn, Yarin Gal, and Sebastian Farquhar. 2023b. Semantic uncertainty: Linguistic invariances for uncertainty estimation in natural language generation. In *The Eleventh International Conference on Learning Representations*.

Yuhang Lai, Chengxi Li, Yiming Wang, Tianyi Zhang, Ruiqi Zhong, Luke Zettlemoyer, Scott Wen tau Yih, Daniel Fried, Sida Wang, and Tao Yu. 2022. Ds-1000: A natural and reliable benchmark for data science code generation. *Preprint*, arXiv:2211.11501.

Alex Lascarides and Nicholas Asher. 2007. Segmented discourse representation theory: Dynamic semantics with discourse structure. In *Computing meaning*, pages 87–124. Springer.

Stephen C. Levinson. 2000. *Presumptive Meanings: The Theory of Generalized Conversational Implicature*. The MIT Press, Cambridge, MA, USA.

Haau-Sing (Xiaocheng) Li, Mohsen Mesgar, André Martins, and Iryna Gurevych. 2023. Python code generation by asking clarification questions. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 14287–14306, Toronto, Canada. Association for Computational Linguistics.

Zhen Lin, Shubhendu Trivedi, and Jimeng Sun. 2024. Generating with confidence: Uncertainty quantification for black-box large language models. *Transactions on Machine Learning Research*.

Alisa Liu, Zhaofeng Wu, Julian Michael, Alane Suhr, Peter West, Alexander Koller, Swabha Swayamdipta, Noah Smith, and Yejin Choi. 2023. We're afraid language models aren't modeling ambiguity. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 790–807, Singapore. Association for Computational Linguistics.

Yue Liu, Chakkrit Tantithamthavorn, Yonghui Liu, and Li Li. 2024. On the Reliability and Explainability of Language Models for Program Generation. *ACM Trans. Software Eng. Method.*, 33(5):1–26.

Anton Lozhkov, Raymond Li, Loubna Ben Allal, Federico Cassano, Joel Lamy-Poirier, Nouamane Tazi, Ao Tang, Dmytro Pykhtar, Jiawei Liu, Yuxiang Wei, Tianyang Liu, Max Tian, Denis Kocetkov, Arthur Zucker, Younes Belkada, Zijian Wang, Qian Liu, Dmitry Abulkhanov, Indraneil Paul, Zhuang Li, Wen-Ding Li, Megan Risdal, Jia Li, Jian Zhu, Terry Yue Zhuo, Evgenii Zheltonozhskii, Nii Osae Osae Dade, Wenhao Yu, Lucas Krauß, Naman Jain, Yixuan Su, Xuanli He, Manan Dey, Edoardo Abati, Yekun Chai, Niklas Muennighoff, Xiangru Tang, Muhtasham Oblokulov, Christopher Akiki, Marc Marone, Chenghao Mou, Mayank Mishra, Alex Gu, Binyuan Hui, Tri Dao, Armel Zebaze, Olivier Dehaene, Nicolas Patry, Canwen Xu, Julian McAuley, Han Hu, Torsten Scholak, Sebastien Paquet, Jennifer Robinson, Carolyn Jane Anderson, Nicolas Chapados, Mostofa Patwary, Nima Tajbakhsh, Yacine Jernite, Carlos Muñoz Ferrandis, Lingming Zhang, Sean Hughes, Thomas Wolf, Arjun Guha, Leandro von Werra, and Harm de Vries. 2024. Starcoder 2 and the stack v2: The next generation. *Preprint*, arXiv:2402.19173.

Qianou Ma, Tongshuang Wu, and Kenneth Koedinger. 2023. Is ai the better programming partner? human-human pair programming vs. human-ai pair programming. *Preprint*, arXiv:2306.05153.

Brielen Madureira, Patrick Kahardipraja, and David Schlangen. 2024. When only time will tell: Interpreting how transformers process local ambiguities through the lens of restart-incrementality. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 4722–4749, Bangkok, Thailand. Association for Computational Linguistics.

Brian McMahan and Matthew Stone. 2020. Analyzing speaker strategy in referential communication. In *Proceedings of the 21th Annual Meeting of the Special Interest Group on Discourse and Dialogue*, pages 175–185, 1st virtual meeting. Association for Computational Linguistics.

Sewon Min, Julian Michael, Hannaneh Hajishirzi, and Luke Zettlemoyer. 2020. AmbigQA: Answering ambiguous open-domain questions. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 5783–5797, Online. Association for Computational Linguistics.

Will Monroe, Robert X.D. Hawkins, Noah D. Goodman, and Christopher Potts. 2017. Colors in context:

A pragmatic neural model for grounded language understanding. *Transactions of the Association for Computational Linguistics*, 5:325–338.

Hussein Mozannar, Valerie Chen, Mohammed Alsobay, Subhro Das, Sebastian Zhao, Dennis Wei, Manish Nagireddy, Prasanna Sattigeri, Ameet Talwalkar, and David Sontag. 2024. The realhumaneval: Evaluating large language models' abilities to support programmers. *Preprint*, arXiv:2404.02806.

Fangwen Mu, Lin Shi, Song Wang, Zhuohao Yu, Binquan Zhang, ChenXue Wang, Shichao Liu, and Qing Wang. 2024. ClarifyGPT: A Framework for Enhancing LLM-Based Code Generation via Requirements Clarification. *Proc. ACM Softw. Eng.*, 1(FSE):2332–2354.

Sydney Nguyen, Hannah McLean Babe, Yangtian Zi, Arjun Guha, Carolyn Jane Anderson, and Molly Q Feldman. 2024. How beginning programmers and code llms (mis)read each other. CHI '24, New York, NY, USA. Association for Computing Machinery.

Ayana Niwa and Hayate Iso. 2024. Ambignlg: Addressing task ambiguity in instruction for nlg. *Preprint*, arXiv:2402.17717.

Konstantinos Papakostas and Irene Papadopoulou. 2023. Model analysis & evaluation for ambiguous question answering. In *Findings of the Association for Computational Linguistics: ACL 2023*, pages 4570–4580, Toronto, Canada. Association for Computational Linguistics.

Brendan Park, Madeline Janecek, Naser Ezzati-Jivan, Yifeng Li, and Ali Emami. 2024. Picturing ambiguity: A visual twist on the Winograd schema challenge. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 355–374, Bangkok, Thailand. Association for Computational Linguistics.

Massimo Poesio and Ron Artstein. 2005. The reliability of anaphoric annotation, reconsidered: Taking ambiguity into account. In *Proceedings of the Workshop on Frontiers in Corpus Annotations II: Pie in the Sky*, pages 76–83, Ann Arbor, Michigan. Association for Computational Linguistics.

Yewen Pu, Kevin Ellis, Marta Kryven, Josh Tenenbaum, and Armando Solar-Lezama. 2020. Program synthesis with pragmatic communication. In *Advances in Neural Information Processing Systems*, volume 33, pages 13249–13259. Curran Associates, Inc.

Yewen Pu, Saujas Vaduguru, Priyan Vaithilingam, Elena Glassman, and Daniel Fried. 2023. Amortizing pragmatic program synthesis with rankings. *Preprint*, arXiv:2309.03225.

Craige Roberts. 2012. Information structure in discourse: Towards an integrated formal theory of pragmatics. *Semantics and Pragmatics*, 5(6):1–69.

Baptiste Rozière, Jonas Gehring, Fabian Gloeckle, Sten Sootla, Itai Gat, Xiaoqing Ellen Tan, Yossi Adi, Jingyu Liu, Romain Sauvestre, Tal Remez, Jérémy Rapin, Artyom Kozhevnikov, Ivan Evtimov, Joanna Bitton, Manish Bhatt, Cristian Canton Ferrer, Aaron Grattafiori, Wenhan Xiong, Alexandre Défossez, Jade Copet, Faisal Azhar, Hugo Touvron, Louis Martin, Nicolas Usunier, Thomas Scialom, and Gabriel Synnaeve. 2024. Code llama: Open foundation models for code. *Preprint*, arXiv:2308.12950.

Advait Sarkar, Andrew D. Gordon, Carina Negreanu, Christian Poelitz, Sruti Srinivasa Ragavan, and Ben Zorn. 2022. What is it like to program with artificial intelligence? *Preprint*, arXiv:2208.06213.

Stefan Schouten, Peter Bloem, Ilia Markov, and Piek Vossen. 2023. Reasoning about ambiguous definite descriptions. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pages 4479–4484, Singapore. Association for Computational Linguistics.

Sander Schulhoff, Michael Ilie, Nishant Balepur, Konstantine Kahadze, Amanda Liu, Chenglei Si, Yinheng Li, Aayush Gupta, HyoJung Han, Sevien Schulhoff, Pranav Sandeep Dulepet, Saurav Vidyadhara, Dayeon Ki, Sweta Agrawal, Chau Pham, Gerson Kroiz, Feileen Li, Hudson Tao, Ashay Srivastava, Hevander Da Costa, Saloni Gupta, Megan L. Rogers, Inna Goncearenco, Giuseppe Sarli, Igor Galynker, Denis Peskoff, Marine Carpuat, Jules White, Shyamal Anadkat, Alexander Hoyle, and Philip Resnik. 2024. The prompt report: A systematic survey of prompting techniques. *Preprint*, arXiv:2406.06608.

Anthony Sicilia, Mert Inan, and Malihe Alikhani. 2024. Accounting for sycophancy in language model uncertainty estimation. *Preprint*, arXiv:2410.14746.

Saujas Vaduguru, Daniel Fried, and Yewen Pu. 2024. Generating pragmatic examples to train neural program synthesizers. In *The Twelfth International Conference on Learning Representations*.

Priyan Vaithilingam, Tianyi Zhang, and Elena L. Glassman. 2022. Expectation vs. Experience: Evaluating the Usability of Code Generation Tools Powered by Large Language Models. In *CHI EA '22: Extended Abstracts of the 2022 CHI Conference on Human Factors in Computing Systems*, pages 1–7. Association for Computing Machinery, New York, NY, USA.

K. van Deemter and S. Peters. 1996. *Semantic Ambiguity and Underspecification*. Center for the Study of Language and Information Publication Lecture Notes. Cambridge University Press.

Anthony J Viera, Joanne M Garrett, et al. 2005. Understanding interobserver agreement: the kappa statistic. *Fam med*, 37(5):360–363.

Bing Wang, Yan Gao, Zhoujun Li, and Jian-Guang Lou. 2023. Know what I don't know: Handling ambiguous and unknown questions for text-to-SQL. In *Findings of the Association for Computational Linguis-*

tics: ACL 2023, pages 5701–5714, Toronto, Canada. Association for Computational Linguistics.

Noah Wang, Zy Peng, Haoran Que, Jiaheng Liu, Wangchunshu Zhou, Yuhan Wu, Hongcheng Guo, Ruitong Gan, Zehao Ni, Jian Yang, et al. 2024. Rolellm: Benchmarking, eliciting, and enhancing role-playing abilities of large language models. In *Findings of the Association for Computational Linguistics ACL 2024*, pages 14743–14777.

Frank Wildenburg, Michael Hanna, and Sandro Pezzelle. 2024. Do pre-trained language models detect and understand semantic underspecification? ask the DUST! In *Findings of the Association for Computational Linguistics: ACL 2024*, pages 9598–9613, Bangkok, Thailand. Association for Computational Linguistics.

L. Williams. 2001. Integrating pair programming into a software development process. In *Proceedings 14th Conference on Software Engineering Education and Training. 'In search of a software engineering profession' (Cat. No.PR01059)*, pages 27–36.

Monika-Zita Zempleni, Remco Renken, John C. J. Hoeks, Johannes M. Hoogduin, and Laurie A. Stowe. 2007. Semantic ambiguity processing in sentence context: Evidence from event-related fMRI. *Neuroimage*, 34(3):1270–1279.

Tianyi Zhang, Tao Yu, Tatsunori Hashimoto, Mike Lewis, Wen-Tau Yih, Daniel Fried, and Sida Wang. 2023. Coder reviewer reranking for code generation. In *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pages 41832–41846. PMLR.

Tong Zhang, Peixin Qin, Yang Deng, Chen Huang, Wenqiang Lei, Junhong Liu, Dingnan Jin, Hongru Liang, and Tat-Seng Chua. 2024. CLAMBER: A benchmark of identifying and clarifying ambiguous information needs in large language models. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 10746–10766, Bangkok, Thailand. Association for Computational Linguistics.

Mingqian Zheng, Jiaxin Pei, Lajanugen Logeswaran, Moontae Lee, and David Jurgens. 2024. When" a helpful assistant" is not really helpful: Personas in system prompts do not improve performances of large language models. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, pages 15126–15154.

Ryszard Zuber. 1973. Presuppositional ambiguity. *Logique et Analyse*, 16(63/64):443–449.

A Generating Dialogue Responses

We simulate our dialogues for plotting code using LLMs and based on the algorithm given in Algorithm 1. In the algorithm, f_D and f_C are defined based on different pragmatics strategies as given in detail in Section §4.2.

Algorithm 1 Dialogue Simulation with LLMs.

```
Require: Problem instance $(u_1^D, I)$
Require: Director model $f_D$
Require: Coder model for NL response $f_C$
Require: Coder model to generate code $g_C$
Require: Number of samples $k$
Require: Number of rounds of dialogue $n$
$S \leftarrow \{s_i \sim g_C(u_1^D) \mid 1 \leq i \leq k\}$
$\mathbf{u} \leftarrow [\,]$
for $n$ times do
    $u^C \leftarrow f_C(\mathbf{u})$
    $\mathbf{u} \leftarrow \mathbf{u} + [u^C]$
    $u^D \leftarrow f_D(I, \mathbf{u})$
    $\mathbf{u} \leftarrow \mathbf{u} + [u^D]$
end for
$c \sim g_C(\mathbf{u})$
return $\mathbf{u}$, $c$
```

Director We prompt the director model f_D to generate instructions and clarifications that guide a coder model toward the correct solution. Since we work with an artificial director agent, we source intents from the DS-1000 dataset. We present the intent to the director in one of two ways – as the code for a reference solution or the plot generated by the code presented as an image. Since a natural language instruction accompanies the DS-1000 problem instances, we seed the interaction using that interaction as the first director turn (u_1^D). We prompt the model to use different strategies to generate responses.

Pragmatic Coder We first extract the code context and the coding instructions from the DS1000 dataset and then convert it into a dialogue format as described in section §4.2. Then, using GPT-4o, we generate codes that respond to the original instruction (sampled k times). To the pragmatic coder, we present a set of possible unique answers it can choose from the generated codes and the dialogue history that is happening and ask for a follow-up utterance for the coder to converge to the solution that the director is describing, i.e. $g_C(u_1^D)$. We then instruct it to give three solutions based on the reasoning types. For the regular director, we provide the reference code (or the reference plot in the case of a multimodal model) and the dialogue history and ask to generate a follow-up utterance to converge to a solution without giving away the answer. All the details of the prompts are given in Appendix B.

Dialogue Policy We employ a rule-based dialogue policy to choose one of the three utterances we generated for each strategy in the simulation. For the first turn of the dialogue, we do not use any LLM generations but directly use the coding instruction from the DS1000 dataset. For the following turns, we generate three different utterances, one for each of the pragmatic director's reasoning ways, and then generate a single utterance without any pragmatic reasoning prompting for the coder for each of the three responses of the director. We use the number of turns as a hyperparameter to generate the dialogue and perform ablation experiments on it. We do not mix reasoning styles across the dialogue's turns, but we choose a single reasoning style for the overall dialogue. We also experiment with providing the reference image or the reference code to the director to see how clarity of instructions affects execution.

B Prompting Details

B.1 Pragmatic Coder

B.1.1 System Prompts:

Director: You are a coding director. There is another coding agent you are going to have a dialogue with. You have a final product in mind. This is going to be named the REF CODE. You want a coder to write the codes for this final product. For the first turn of the dialogue, you give a specific instruction or a question about the final product. Then, the coder will give you some answers, and then you will have another turn to refine the codes.

Coder: You are a coding agent. There is another director agent you are going to have a dialogue with. The things you say depend on your persona. You have the following different personas (reasoning styles):

- Cooperative Persona (Pragmatic): You want to converge on the solution as quickly as possible and follow Grice's Maxims when choosing your words. You anticipate the director's cooperative reasoning. You possess theory-of-mind capabilities and common sense.

- Discourse Reasoning Persona: Everything you say is connected to the previous turn with a relation. The possible discourse relations are Comment, Clarification Question, Elaboration, Acknowledgment, Continuation, Explanation, Conditional, Alternation, Result, Background, Narration, Correction, Parallel, Contrast. You try to identify the relation between the utterance of the director in the previous with your utterance. Then you reply with an utterance that has the appropriate relation.

- Questioning Persona: Everything you say has an implicit question underneath it. You should tell what the director is actually asking for (the question under their instruction), and give your answer to that implicit question.

The director has a final product in mind. You, as the coder, write the codes for this final product or have a dialogue about the instruction. For the first turn of the dialogue, the director gives a specific instruction or a question about the final product. Then, you will give some answers, and then the director will have another turn to refine the codes.

user prompts:

Director: REF CODE: "'+ ref-code "' + DIALOGUE HISTORY:" + dialogue-history + What can you say on the follow-up turn for the coder to converge to the reference code? Do not mention anything about the REF CODE, and don't give away the answer.

Coder: POSSIBLE GENERATED CODES: Solution 1: "'CODE"' Solution 2: "'CODE"' ....

DIALOGUE HISTORY: + dialogue-history + What can you say on the following turn as the coder to converge to the solution that the director has in mind? Give responses for all types of your personas. Personas must not give the same solution! Your solution MUST NOT contain any new code. You can talk about the provided code.

C Additional Experiments with Various Models

This section presents results from several experiments with multiple other models, such as CodeLLaMA, LLaMMA3.2, StarCoder-2 in Table 5 and Table 6.

D Abstract Syntax Tree (AST) Functional Uniqueness Algorithm

In this section, we detail the AST-based function uniqueness comparison algorithm between two separate generated functions. The code for the algorithm is given in Listing 1. We find this form of comparison to be appropriate for plotting tasks as the lines of code of interest are generally the calls

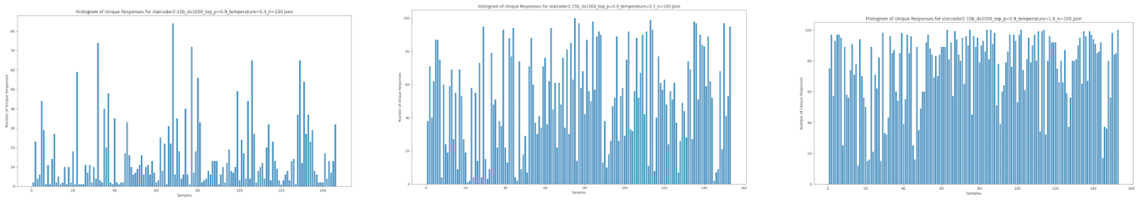


Figure 5: This figure shows the change in unique responses of code completions depending on the temperature of the model. From the left, the plots are showing histograms for 0.3, 0.7, and 1.0 temperatures. The horizontal axis is the question number from the DS1000 matplotlib dataset. It is observable that the uniqueness is high for higher temperatures, expectedly. However, very high temperatures may have minor differences that increase the overall uniqueness. Hence, a moderate temperature like 0.7 gives more reliable results for further experimentation.

			Pass@1↑	d_s↓
Baseline		No Dialogue	0.422	0.744
Pragmatic Coder	with code	Cooperative	0.427	0.640
		Discoursive	0.467	0.613
		Inquisitive	0.396	0.716
	with image	Cooperative	0.447	**0.584**
		Discoursive	**0.493**	0.624
		Inquisitive	0.393	0.711

Table 5: This figure shows the main results of our experimentation for CodeLLaMA as the coder, and GPT-4o as the director, and the baseline corresponds to StarCoder2. Here, we give the metrics for both executability and sampling diversity. Having a dialogue generally performs better than the baseline code completion without any dialogue. For each pragmatic setting, we experiment with all the reasoning styles and have an image or code as the reference solution for the director.

to library functions, particularly those provided by the `matplotlib` API.

E Temperature Adjustments

We present our experimentation results for the temperature tuning in Figure 5.

	No Dialogue			
Pass @ 1	**OG**	**I**	**C**	**U**
GPT 4o	68.38%	68.38%	87.74%	81.29%
LLaMA 3.2	63.23%	64.52%	77.42%	65.81%
LLaMA → GPT	-	66.45%	85.16%	78.71%
GPT → LLaMA	-	77.42%	81.29%	79.36%

Table 6: This table shows the results for ambiguity representation transfer between different models. OG: original prompt, I: image reprompt, C: code reprompt, U: unit test reprompt.

F Example Dialogues

We present the reference-image-based coder-director dialogues for the final example from Table 4.

```python
def compare_parse_trees(response1, response2):
    """Compare the parse trees of two responses."""
    unique_function_calls = []
    unique_params = {}
    unique_keywords = {}
    try:
        tree1 = ast.parse(response1)
        functions1 = get_params(tree1)
        tree2 = ast.parse(response2)
        functions2 = get_params(tree2)
        for function in functions1.keys():
            if function not in functions2.keys():
                unique_function_calls.append(function)
            else:
                for i, arg in enumerate(functions1[function]):
                    if arg not in functions2[function]:
                        if function not in unique_params.keys():
                            unique_params[function] = []
                        unique_params[function].append(arg)
                    if isinstance(arg, dict):
                        for key in arg.keys():
                            for j in range(len(functions2[function])):
                                if isinstance(functions2[function][j], dict):
                                    if key not in functions2[function][j].keys():
                                        if function not in unique_keywords.keys():
                                            unique_keywords[function] = []
                                        unique_keywords[function].append(key)
                                    else:
                                        if arg[key] != functions2[function][j][key]:
                                            if function not in unique_keywords.keys():
                                                unique_keywords[function] = []
                                            unique_keywords[function].append(key)
    except SyntaxError:
        print("Syntax Error")
    return unique_function_calls, unique_params, unique_keywords
```

Listing 1: This code snippet shows how the functions of two separate generated codes are compared using their ASTs.

Coding Question: highlight in red the x range 2 to 4 (0.667 Initial Pass@1)

Cooperative Dialogue (0.967 Final Pass@1)

Director: highlight in red the x range 2 to 4

Coder: Both solutions will highlight the x range from 2 to 4 in red, but they differ in transparency. Solution 1 uses an ‘alpha‘ of 0.3, which means it’s more transparent than Solution 2 with an ‘alpha‘ of 0.5. If you want a less noticeable highlight, go with Solution 1; for more prominence, choose Solution 2.

Director: Increase the opacity of the highlighted area so that it’s fully opaque instead of transparent. Make sure the highlight is prominently visible.

Coder: To make the highlight fully opaque, you’ll need to set the ‘alpha‘ parameter to 1. You can modify either of the existing solutions by changing ‘alpha=0.3‘ or ‘alpha=0.5‘ to ‘alpha=1‘. This will eliminate transparency entirely and ensure that the red highlight is prominently visible.

Director: Make sure the highlight completely covers the x range from 2 to 4 with no gaps or transparency. The highlighted area should be clearly distinguished and fully filled in red.

Discoursive Dialogue (1.0 Final Pass@1)

Director: highlight in red the x range 2 to 4

Coder: Clarification Question – The provided solutions differ in their transparency settings through the ‘alpha‘ values. Could you specify your preference regarding how prominent or subtle you’d like this red highlighting effect to be?

Director: Please ensure that the red highlight is fully opaque, with no transparency effect.

Coder: Correction – To make sure we match your requirement of having no transparency, you’ll need to modify the ‘alpha‘ value in either solution from its current setting to ’1’, achieving a fully opaque appearance.

Director: Please ensure the x range highlighted in red is fully opaque with no transparency. We need it to stand out distinctly against the rest of the plot.

Inquisitive Dialogue (0.167 Final Pass@1)

Director: highlight in red the x range 2 to 4

Coder: What you seem to want is ensuring that the area between x=2 and x=4 is clearly visible on your plot highlighted in red. Do you want it to be very noticeable or somewhat subtle? The difference here is that one solution is more transparent than the other; which visibility level do you prefer?

Director: Please make the highlighted area more noticeable by removing any transparency so it’s fully opaque.

Coder: The implicit question here is about achieving complete opacity for the highlighted area between x=2 and x=4 on a plot. If neither existing solution (‘Solution 1‘ or ‘Solution 2‘) meets your requirements—since they include some degree of transparency—you’ll need an alternative with no alpha setting or perhaps adjusting ’alpha’ to ‘1‘.

Director: Ensure the red highlighted area between x=2 and x=4 is fully opaque without any transparency. The ‘alpha‘ parameter should be set to 1, or simply not included at all to achieve full opacity.