

Iterative Resolution of Prompt Ambiguities Using a Progressive Cutting-Search Approach

Fabrizio Marozzo¹[0000–0001–7887–1314]

University of Calabria, Italy
fmarozzo@dimes.unical.it

Abstract. Generative AI systems have revolutionized human interaction by enabling natural language-based coding and problem solving. However, the inherent ambiguity of natural language often leads to imprecise instructions, forcing users to iteratively test, correct, and resubmit their prompts. We propose an iterative approach that systematically narrows down these ambiguities through a structured series of clarification questions and alternative solution proposals—illustrated with input/output examples as well. Once every uncertainty is resolved, a final, precise solution is generated. Evaluated on a diverse dataset spanning coding, data analysis, and creative writing, our method demonstrates superior accuracy, competitive resolution times, and higher user satisfaction compared to conventional one-shot solutions, which typically require multiple manual iterations to achieve a correct output.

Keywords: Ambiguity Resolution · Interactive Prompting · Generative AI · Prompt Engineering · Natural Language Processing

1 Introduction

The rapid adoption of generative AI systems is fundamentally reshaping how humans interact with technology. Widely used models from leading IT companies, such as OpenAI’s GPT-4 and other cutting-edge solutions developed by companies like Google and DeepMind, have demonstrated advanced reasoning capabilities and the ability to articulate their thought processes during interactions [2,24]. This enables users to engage with technology through natural language, seamlessly solving a wide array of tasks—including creative writing and data analysis—in an intuitive and conversational manner [9]. By not only delivering answers but also providing insights into their reasoning, these models empower users of all skill levels to work more efficiently and make better-informed decisions.

In the field of coding, these systems have become invaluable. They not only assist programmers in writing and understanding code but also help diagnose and resolve errors [6]. This shift is leading to a new paradigm: strong declarative programming through natural language prompts. Traditionally, declarative programming has involved using formal languages to specify what a program should

accomplish without detailing the underlying control flow. Languages like SQL for database queries, HTML/CSS for web layouts, and various high-level configuration languages exemplify this approach—users define the desired outcome, and the system figures out the steps to achieve it [14]. However, these languages require users to learn specific syntaxes and semantics, which can be a barrier for many.

Building on this shift, AI-driven systems are now making coding even more accessible by allowing users to articulate their intent in everyday language. This natural language approach democratizes programming by reducing the steep learning curve associated with formal languages [20,18]. The AI interprets the user’s intent and translates these prompts into executable code, enabling both programmers and non-programmers to harness the power of automation more intuitively and efficiently, thereby accelerating development cycles and making technology more accessible [4,19]. However, this convenience introduces a significant challenge: natural language is inherently imprecise and fraught with ambiguities. Unlike a formal algorithm—defined by a clear, unambiguous list of instructions—natural language can be interpreted in multiple ways [23]. As anyone with programming experience knows, ambiguous requirements often necessitate an iterative process of refinement: a provisional solution is provided, the user tests it, and subsequent adjustments are made until the final, correct implementation is achieved [11].

To address this challenge, we propose an iterative approach that systematically resolves ambiguities through a structured series of clarification questions. The process begins by analyzing the user’s input to identify potential ambiguities. For each issue detected, the system engages the user in a dialogue, presenting alternative solutions—often illustrated with input/output examples—to clarify the intended meaning. As ambiguities are resolved, the system dynamically adjusts the remaining uncertainties. If resolving one ambiguity clarifies or eliminates others, they are automatically removed from the process. This iterative refinement continues until all ambiguities have been addressed. At this stage, the system generates a final, precise solution that accurately reflects the user’s intent. To further validate the solution, the system provides representative examples, including edge cases, to illustrate the main behavior of the code. These examples allow users to assess how the final solution performs under various conditions, ensuring that the results align with their expectations.

To evaluate the effectiveness of our iterative ambiguity resolution process, we conducted a study using a diverse dataset of natural language prompts across three domains: coding, data analysis, and creative writing. We assessed the system’s ability to accurately identify and resolve ambiguities, comparing it to conventional large language models that generate one-shot solutions without interactive clarification. Our evaluation focused on three key aspects: accuracy, efficiency, and user experience. The results demonstrate that our iterative approach significantly improves the precision of generated outputs by systematically refining ambiguous prompts, ensuring that the final solution better aligns with user intent. Additionally, despite requiring an initial clarification phase, our

method maintains competitive resolution times by reducing the number of failed attempts and revisions typically needed in one-shot prompting. User feedback further highlights the advantages of guided disambiguation, reporting higher satisfaction due to improved clarity and reduced trial-and-error effort.

This paper is structured as follows: Section 2 reviews related work on ambiguity resolution and prompt engineering. Section 3 details our proposed iterative solution based on a progressive cutting-search approach. Section 4 presents our experimental results, including representative examples and evaluations. Finally, Section 5 concludes the paper by summarizing our contributions and outlining directions for future research.

2 Related Work

Recent advances in large-scale language models (LLMs) have revolutionized multiple domains by demonstrating remarkable capabilities in natural language processing tasks, including coding [16], data analysis [21], and creative writing [12]. Systems such as GPT-4, Claude, Gemini, and other state-of-the-art large language models leverage massive datasets and complex architectures to generate contextually relevant, coherent, and often insightful outputs across diverse applications. These models not only deliver high-quality responses but also provide explanations of their reasoning processes, thereby enhancing interactive experiences between humans and machines [5,8].

In the realm of software development, the code generation capabilities of LLMs have had a profound impact. Tools such as OpenAI Codex (powering GitHub Copilot), Amazon CodeWhisperer, Google AlphaCode, DeepMind AlphaDev, and Tabnine assist developers in writing, understanding, and debugging code. These systems leverage advanced AI models to generate multi-line functions, optimize existing code, and suggest improvements, streamlining the development process and reducing the entry barrier for new programmers [17].

For example, Chen et al. [6] provide empirical evidence that LLMs are capable of producing coherent and functional code segments tailored to specific requirements, thereby reducing the manual effort typically needed for debugging and refinement. Li et al. [15] propose a method that leverages prompt engineering to rapidly generate source code, demonstrating that dynamically optimized prompts can significantly enhance the consistency and quality of generated code. Choi et al. [7] further explore how prompt-based approaches can improve code comprehension and ensure that the intended functionality is preserved, addressing common issues like ambiguity in code behavior. Additionally, Wang et al. [22] show that prompt tuning can outperform traditional fine-tuning methods, particularly in low-resource scenarios, by aligning model outputs more closely with developer expectations. Collectively, these studies underscore the transformative role of LLMs and prompt engineering in modern software development, paving the way for more efficient, reliable, and accessible coding practices.

Ambiguity in natural language poses significant challenges to machine understanding, particularly in tasks such as code generation and data analysis where

precision is critical. Recent studies have explored iterative disambiguation techniques that engage users in clarification dialogues, thereby systematically resolving ambiguities. These approaches emphasize the importance of interactive systems that adapt to user feedback and progressively narrow down multiple interpretations until a precise and unambiguous instruction is achieved [3]. For example, He et al. [13] introduced a human–machine co-adaptation framework that leverages multi-turn dialogues and targeted clarifying questions to iteratively refine ambiguous prompts, ensuring that the intended meaning is progressively clarified. Similarly, Aina and Linzen [1] investigated how language models maintain and adjust syntactic uncertainty by sampling multiple completions, effectively quantifying the extent of ambiguity in a prompt. Moreover, best practices in prompt engineering, as outlined by Sabit Ekin [10], advocate for iterative prompt refinement and the explicit specification of constraints to mitigate vagueness and improve output quality. Collectively, these studies underscore that integrating interactive, iterative disambiguation processes is essential to enhance the precision and contextual alignment of model responses across a variety of applications.

Unlike conventional approaches that rely on trial-and-error refinements, our method introduces a structured iterative clarification framework that systematically resolves ambiguities before generating a final response. Rather than producing an initial output that may require multiple manual corrections, our approach engages users in a guided resolution process, dynamically eliminating invalid interpretations and refining intent through targeted clarification steps. This ensures that the final output is both precise and aligned with user expectations while minimizing the need for post-hoc adjustments and reducing overall interaction time.

3 Proposed approach

The proposed approach is aimed at transforming ambiguous natural language prompts into precise, executable solutions by systematically identifying and resolving uncertainties. It comprises three distinct phases: (i) *detecting ambiguities in the prompt*; (ii) *iteratively resolving these ambiguities through a decision-tree dialogue with the user*; and (iii) *generating and validating the final solution with representative examples, including edge cases*. In the following, we provide a detailed description of the main steps of our approach, whose execution flow is depicted in Figure 1.

In the initial phase—*identifying ambiguities in the prompt*—the system conducts a thorough analysis of the user’s natural language input to pinpoint potential sources of misunderstanding. By leveraging advanced natural language processing techniques, including a chatbot such as GPT-4o via API, it detects ambiguous terms and phrases that could lead to multiple interpretations. Additionally, the system considers contextual information provided by the user, such as domain-specific constraints, to refine the ambiguity detection process. This allows the system to differentiate between inherently vague expressions and

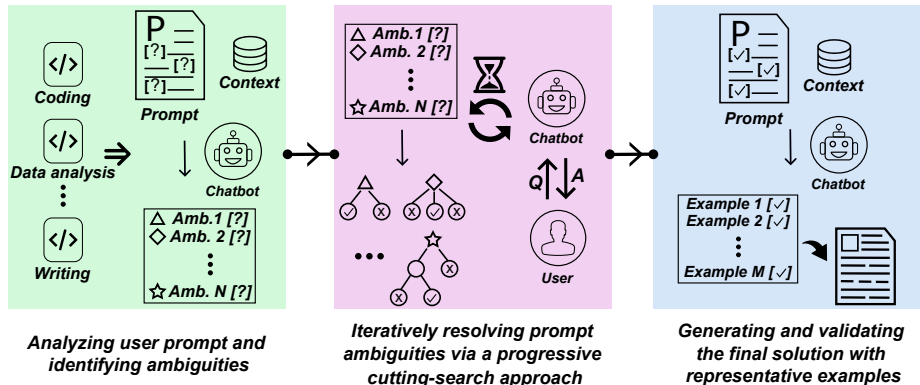


Fig. 1. Execution flow of the proposed approach.

terms that may be clear within a specific context. By incorporating both linguistic analysis and contextual cues, this step lays the foundation for a more precise and targeted resolution process.

The second phase focuses on *iteratively resolving these ambiguities via a progressive cutting-search approach*. Here, the system engages the user in a structured dialogue using a chatbot model to present alternative interpretations for each detected ambiguity, along with illustrative input/output examples. The dialogue follows a progressive cutting-search strategy, in which each user response eliminates invalid interpretations, dynamically refining the search space until a precise and unambiguous meaning is reached. This iterative narrowing process ensures that every uncertainty is addressed while minimizing unnecessary interactions. By guiding the system through successive clarifications, users efficiently arrive at a well-defined prompt that accurately reflects their intent.

In the final phase—*generating and validating the final solution with representative examples*—the system compiles the resolved interpretations into a definitive solution, ensuring that all ambiguities have been systematically addressed. To further validate the accuracy and robustness of the output, the system, again leveraging a chatbot generates representative examples, including edge cases and borderline scenarios, that illustrate the main behavior of the final solution. These examples allow users to assess whether the solution meets their specific requirements, performs consistently under different conditions, and aligns with their expectations. If necessary, users can provide additional feedback to refine the solution before finalization, ensuring a high degree of reliability and adaptability across various contexts.

4 Experimental Results

In this section, we present the experimental results of our iterative ambiguity resolution system. Our evaluation assesses the system’s effectiveness across di-

verse domains, comparing its performance to standard one-shot methods. We report both quantitative metrics—including ambiguity detection accuracy, resolution time, and user satisfaction—and qualitative analyses of representative outputs. The results demonstrate that our guided, interactive approach not only improves the precision of generated outputs but also significantly reduces the time and effort required from users.

Further details on these findings are provided in Section 4.1, which presents specific examples of ambiguity resolution across different domains, and Section 4.2, which offers an overview of the system’s performance based on aggregated results and key evaluation metrics.

4.1 Case Studies and Input Examples

As a first case study, we consider a generic transactional database commonly used in e-commerce systems. This database consists of two primary tables: `orders` and `customers`. The `orders` table contains fields such as `order_id`, `order_date`, `customer_id`, `total_amount`, and `order_status`, capturing essential details about each transaction. The `customers` table includes fields like `customer_id`, `name`, `email`, `registration_date` and `total_spent`, providing information about customer identity, activity, and purchasing behavior.

Below is an example of a user-provided prompt that will be examined for ambiguities when converting natural language into SQL. This prompt serves as the starting point of our process.

Prompt: Translate the following request into SQL: *Find all orders placed last month by customers with high spending habits.*

The next box lists the ambiguities detected in the prompt. These ambiguities indicate the areas that require further clarification.

Ambiguities:
A1: Time Frame Interpretation (“*last month*”);
A2: Definition of “*high spending habits*”.

Next, we present a sequence of clarifying questions designed to resolve these ambiguities. Each step in this dialogue helps narrow down the possible interpretations.

Questions:
Step 1: Clarifying the Time Frame
Q1: How should we interpret “*last month*” in the query?
Option A: Use the previous calendar month.
Option B: Use the last 30 days relative to today.
Step 2: Defining “*High Spending Habits*”
Q2: How should we determine if a customer has high spending habits?

Option A: Consider customers whose total spending exceeds a specific monetary threshold.

Option B: Use an existing flag or field (e.g., a "VIP" status) in the customer database.

Step 3: If *Option A* is selected for *Q2*, then:

Q3: Would you like to specify the spending threshold?

Option A1: Yes, let me input a threshold value.

Option A2: No, use a default threshold (e.g., \$1,000).

Suppose the user selects *Option A* for *Q1* (previous calendar month) and *Option A* for *Q2* with a threshold of \$1,000 (*Option A1*). The final SQL query might then be:

```

1 SELECT o.order_id, o.order_date, c.customer_id, c.total_spent
2 FROM orders o JOIN customers c ON o.customer_id =
   c.customer_id
3 WHERE o.order_date BETWEEN '2025-02-01' AND '2025-02-28'
4 AND c.total_spent > 1000;
```

The following box shows representative examples of the query's behavior.

Example Outputs:

Selected: Customer with `customer_id` 101 has total spending \$1,200, and an order on 2025-02-15.

Not Selected: Customer with `customer_id` 102 has total spending \$950, even if the order is within **February** 2025.

Not Selected: Customer with `customer_id` 103 has total spending \$1,500, but the order was placed on 2024-12-28.

This example demonstrates how the initial ambiguous prompt is transformed into a precise SQL query, and how the final query behaves with different customer data scenarios.

As a second example, we consider a dataset containing daily temperature readings. The task is to write a Python function that calculates the average temperature while excluding outlier values. The dataset is provided as a list of numeric temperature readings, and outliers should be identified using a statistical method.

Below is an example of a user-provided prompt that will be examined for ambiguities when converting natural language into a Python function. This prompt serves as the starting point of our process.

Prompt: Write a Python function that calculates the average temperature from a list of readings, excluding outliers.

The following box lists the ambiguities detected in the prompt. These ambiguities indicate areas that require further clarification.

Ambiguities:

A1: *Outlier Definition:* What criteria should be used to determine outliers (e.g., using the IQR method, z-score, or a fixed threshold)?

A2: *Statistical Basis:* Should outliers be determined relative to the median or the mean of the dataset?

A3: *Empty Result Handling:* What should the function return if all readings are identified as outliers?

Next, we present a sequence of clarifying questions designed to resolve these ambiguities. Each step in the dialogue helps narrow down the possible interpretations.

Questions:

Step 1: Defining Outliers

Q1: Which statistical method should be used to identify outliers?

Option A: Use the IQR method.

Option B: Use the z-score method.

Step 2: Determining Statistical Basis

Q2: Should outliers be determined relative to the median or the mean?

Option A: Use the median.

Option B: Use the mean.

Step 3: Handling Empty Results

Q3: What should the function return if all readings are excluded as outliers?

Option A: Return `None`.

Option B: Return 0.

Suppose the user selects *Option A* for *Q1* (IQR method), *Option A* for *Q2* (use the median), and *Option A* for *Q3* (return `None`). The final Python function might then be:

```

1 import numpy as np
2 def average_temperature(readings):
3     if not readings:
4         return None
5     # Calculate the first and third quartiles based on the median
6     q1 = np.percentile(readings, 25)
7     q3 = np.percentile(readings, 75)
8     iqr = q3 - q1
9     lower_bound = q1 - 1.5 * iqr
10    upper_bound = q3 + 1.5 * iqr
11    # Filter out outliers
12    filtered = [temp for temp in readings if lower_bound <= temp <=
    ↪    upper_bound]
```

```
13     if not filtered:
14         return None
15     return sum(filtered) / len(filtered)
```

The following box shows representative examples of the function’s behavior:

Example Outputs:

Selected: For readings [32, 35, 36, 38, 120], assuming 120 is an outlier, the function returns an average of 35.25.

Not Selected: For readings [15, 16, 15, 1000], if 1000 is identified as an outlier, the average is computed from [15, 16, 15].

Empty Result: For readings [1000, 1020, 980], if all values are considered outliers, the function returns None.

4.2 Comprehensive Evaluation

We conducted a comprehensive evaluation of our iterative ambiguity resolution approach using a testing set that covers a range of ambiguous prompts from different domains. All input data, including the original prompts with ambiguities, the identified ambiguities, and the corresponding disambiguated prompts, are available at <https://github.com/SCAlabUnical/PromptAmbiguityDataset/>. This diverse testing set was designed to assess the system’s performance in transforming vague, natural language instructions into precise and executable prompts.

Our evaluation is organized around three case studies:

1. **Coding:** The system transforms ambiguous natural language requests into precise code (e.g., SQL, Python) by iteratively clarifying uncertainties regarding parameters, algorithms, and data handling. This results in executable code that faithfully meets the intended requirements.
2. **Data Analysis:** Ambiguous instructions for data analysis are refined into clear, well-defined tasks through a guided dialogue. This process resolves uncertainties about statistical measures and selection criteria, yielding robust, executable analytical scripts.
3. **Creative Writing:** The system refines generic writing prompts—lacking details on characters, setting, or tone—into detailed instructions. Through iterative clarification, it enables the generation of coherent, creative narratives that align with the user’s intent.

Our evaluation compares our iterative ambiguity resolution process with a standard one-shot output generation system. In the standard approach, users typically receive a single output from their initial prompt—often a piece of code or analysis—and then try the result; if it is not as expected, they must manually revise and resubmit new prompts to correct errors or address misunderstandings.

This cycle of testing and prompt adjustment continues until the desired outcome is achieved. By contrast, our iterative process guides the user through targeted clarifications and refinements in a structured dialogue, streamlining the path to an accurate final output. As a chatbot system, we have chosen to use GPT-4o (accessed via our API) for generating responses in our iterative process and the standard one-shot version through a chat interface, although our approach is designed to be easily adaptable to other similar systems.

To thoroughly evaluate the performance and user impact of our iterative ambiguity resolution process, we focus on three key aspects:

1. **Ambiguity Identification Accuracy:** We measure how effectively the system detects ambiguous elements within user prompts by calculating precision and recall against expert annotations. A higher F1-score indicates that our system reliably identifies potential ambiguities.
2. **Time Savings and Overall Productivity:** We compare the total user interaction time—from initial prompt submission to final output acceptance—as well as the number of manual corrections required by our guided approach versus the standard one-shot method. A reduction in these metrics indicates improved efficiency and productivity.
3. **Interactive Ambiguity Resolution Efficiency:** We assess user satisfaction with the guided ambiguity resolution process through questionnaires that evaluate the clarity of the generated output, the efficiency of the dialogue, and overall satisfaction with the iterative refinements. This provides insights into the benefits of structured disambiguation over conventional methods.

For our evaluation, we engaged ten human evaluators with expertise in coding, mathematical reasoning, and statistical analysis to ensure accurate interpretation and assessment of ambiguities. The dataset consists of 75 ambiguous prompts, evenly distributed across the three case studies (25 for coding, 25 for data analysis, and 25 for creative writing). The number of ambiguities per prompt is limited and varies between 1 and 5. Each evaluator was assigned ten queries, selected randomly across different case studies, ensuring diversity in prompt evaluation. Their domain knowledge enables a reliable assessment of both the system’s ability to detect ambiguities and the effectiveness of its resolution, ensuring that the generated outputs are both precise and contextually relevant.

In the following subsections, we present the detailed results for each of these evaluation criteria, demonstrating the effectiveness of our approach through empirical analysis and user feedback.

4.2.1 Ambiguity Identification Test Using our dataset, we evaluate the clarity of the ambiguities identified by our system by comparing its results against expert annotations. For each ambiguous prompt, reference ambiguities—defined and inserted into the dataset by human experts—serve as reference. The system detects a set of ambiguities for each prompt, and we compute

the intersection between the system’s findings and the reference. Precision is calculated as the ratio of the intersection (i.e., correctly identified ambiguities) to the total ambiguities detected by the system, while recall is the ratio of the intersection to the total ambiguities present in the reference. The F1 score is then derived as the harmonic mean of precision and recall. Note that any additional ambiguities detected by the system that are not present in the reference are not included in these calculations, even though they may be interesting and timely. Table 1 summarizes the performance across three use cases: coding, data analysis, and creative writing.

In our evaluation, the system achieves high performance in the coding domain with a precision of 0.84, recall of 0.87, and F1 score of 0.85, reflecting the clear and structured nature of programming languages. In data analysis, the system shows a precision of 0.77, recall of 0.87, and F1 score of 0.82, indicating that it generally captures expert-identified ambiguities, though sometimes it flags additional ones. In creative writing, performance is lower (precision 0.74, recall 0.64, F1 0.69), highlighting the challenge of detecting subtle ambiguities in free-text. Overall, these results demonstrate the system’s strength in structured contexts and the need for further refinement for free-text writing.

Use Case	Precision	Recall	F1-Score
Coding	0.84	0.87	0.85
Data Analysis	0.77	0.87	0.82
Creative Writing	0.74	0.64	0.69

Table 1. Ambiguity Identification Performance

4.2.2 Time Efficiency Test We measure the time users spend resolving ambiguities with our iterative approach, including the writing of the initial prompt, the system’s detection of ambiguities, and the interactive clarification process between the user and the system. This process concludes with the generation of a final, corrected result. If the output is not entirely accurate, any additional refinements beyond the guided interaction are handled independently through further interactions with ChatGPT-4o. To provide a comparative analysis, we also evaluate the total number of interactions and the total time required in a standard one-shot approach until the correct output is achieved. Our evaluation is based on 10 randomly selected tests for each use case—coding, data analysis, and creative writing—to ensure a representative assessment. The aggregated averages of these results are presented in Table 2.

The results show that our iterative approach significantly reduces the number of interactions across all tasks, with coding and data analysis requiring approximately half as many interactions as the standard approach. This reduction is also reflected in the average time spent, where users complete tasks faster using the iterative method. In coding and data analysis, time savings are particularly notable, with reductions of approximately 50% and 40%, respectively. In creative

Task Type	Avg. number of Interactions		Avg. time (minutes)	
	Standard	Iterative	Standard	Iterative
Coding	6.1	3.0	17.9	9.0
Data Analysis	5.4	3.5	18.3	10.9
Creative Writing	7.2	5.8	13.8	10.3

Table 2. Comparison of interactions and average user time between a standard one-shot approach and our iterative ambiguity resolution approach for three case studies.

writing, while the iterative approach still leads to a lower number of interactions and time spent, the reduction is less pronounced, likely due to the subjective nature of writing tasks, where refinement often requires more iteration. These findings highlight the efficiency of our approach in minimizing user effort while improving task completion time.

4.2.3 Interactive Resolution Test For each ambiguous query, we measure how many clarification iterations are needed. In one test, our system might require from one to five rounds of dialogue to resolve ambiguities, resulting in a final, correct query. In contrast, a standard system might output an incorrect query that forces the user to manually debug and revise the query over multiple attempts. User feedback is collected via questionnaires to assess clarity and satisfaction with the iterative process.

To better understand the user experience, the following questions were posed:

1. How clear was the final query generated by the system?
2. How efficient was the dialogue process in resolving ambiguities?
3. How satisfied are you with the overall iterative process?
4. How likely are you to recommend this system to others based on the clarification process?
5. To what extent did the iterative clarifications improve the accuracy of the final query?

Each question was rated on a scale of 1 to 5, with 5 being the most favorable.

Satisfaction Question	Avg. Rating (out of 5)
Clarity of the Final Query	4.4
Efficiency of the Clarification Process	4.8
Overall Satisfaction with the Dialogue	4.3
Likelihood to Recommend the System	4.6
Perceived Improvement from Iteration	4.1

Table 3. User satisfaction ratings for the iterative ambiguity resolution process.

Table 3 presents user satisfaction ratings for the iterative ambiguity resolution process. The results indicate that users found the clarification process highly

efficient (4.8), suggesting that the guided approach effectively refines ambiguous prompts. The clarity of the final query (4.4) and likelihood to recommend the system (4.6) further highlight the perceived usefulness and reliability of the method. The overall satisfaction with the dialogue (4.3) reflects positive user experience, while the perceived improvement from iteration (4.1) suggests that users recognize the benefits of interactive disambiguation, though with slightly lower enthusiasm compared to other aspects. These findings demonstrate that users appreciate the structured resolution approach, reinforcing its value as an alternative to conventional one-shot methods.

5 Conclusion

The findings of our work demonstrate that a guided, stateful approach to prompt disambiguation significantly enhances the performance and usability of generative AI systems. By engaging users in an iterative dialogue to clarify ambiguous prompts, our method consistently yields more accurate outputs and streamlines the overall problem-solving process. This structured guidance reduces the reliance on extensive post-generation corrections, ultimately saving valuable time and effort compared to standard, free-form prompting techniques. Looking ahead, future work will focus on integrating this guided prompt methodology into a broader range of applications. We plan to explore its potential to endow generative AI systems with enhanced stateful capabilities, thereby further improving their effectiveness and user-friendliness in real-world scenarios.

Acknowledgements

This work was supported by the research project “INSIDER: INtelligent Ser-vIce Deployment for advanced cloud-Edge integRation” granted by the Italian Ministry of University and Research (MUR) within the PRIN 2022 program and European Union - Next Generation EU (grant n. 2022WWSCRR, CUP H53D23003670006).

References

1. Aina, L., Linzen, T.: The language model understood the prompt was ambiguous: Probing syntactic uncertainty through generation. arXiv:2109.07848 (2021)
2. Akhtar, Z.B.: Unveiling the evolution of generative ai (gai): a comprehensive and investigative analysis toward llm models (2021–2024) and beyond. *Journal of Electrical Systems and Information Technology* **11**(1), 22 (2024)
3. Berg, J., Smith, A., Patel, R.: Resolving ambiguities in natural language instructions: An interactive approach. In: *Proceedings of the Conference on Empirical Methods in Natural Language Processing* (2019)
4. Bommasani, R., Hudson, D.A., Adeli, E., Altman, R., Arora, S., von Arx, S., Bernstein, M., Bohg, J., Boutellier, R., Chang, K., et al.: On the opportunities and risks of foundation models. arXiv:2108.07258 (2021)

5. Brown, T.B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., et al.: Language models are few-shot learners. In: *Advances in Neural Information Processing Systems* (2020)
6. Chen, M., Tworek, J., Jun, H., Yuan, Q., Pinto, H.P.d.O., Kaplan, J., et al.: Evaluating large language models trained on code. *arXiv:2107.03374* (2021)
7. Choi, H., Park, H., Choi, Y.J., Han, K.: Consistency of code: A prompt based approach to comprehend functionality. *Proceedings of the Asia-Pacific Software Engineering Conference* (2023)
8. Chowdhery, A., Narang, S., inston, J., et al.: Palm: Scaling language modeling with pathways. In: *Proc. of the Int. Conf. on Machine Learning* (2022)
9. Devlin, J., Chang, M.W., Lee, K., Toutanova, K.: Bert: Pre-training of deep bidirectional transformers for language understanding. In: *Proc. of NAACL-HLT* (2019)
10. Ekin, S.: Prompt engineering for chatgpt: A quick guide to techniques, tips, and best practices (2023)
11. Elsen, P.: *From Prompts to Programs: Enhancing Creative Coding with AI*. Elsevier (2022)
12. Fan, A., Lewis, M., Dauphin, Y.: Hierarchical neural story generation. In: *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. pp. 889–898 (2018)
13. He, Y., Wang, J., Li, K., Wang, Y., Sun, L., Yin, J., Zhang, M., Wang, X.: Enhancing intent understanding for ambiguous prompts through human-machine co-adaptation. *arXiv:2501.15167* (2025)
14. Lee, K.: *Declarative Programming Paradigms: A Survey*. Springer (2020)
15. Li, Y., Shi, J., Zhang, Z.: An approach for rapid source code development based on chatgpt and prompt engineering. *IEEE Access* (2024)
16. Liu, J., Xia, C.S., Wang, Y., Zhang, L.: Is your code generated by chatgpt really correct? rigorous evaluation of large language models for code generation. *Advances in Neural Information Processing Systems* **36** (2024)
17. Nguyen-Duc, A., Cabrero-Daniel, B., Przybylek, A., Arora, C., Khanna, D., Herda, T., Rafiq, U., Melegati, J., Guerra, E., Kemell, K.K., et al.: Generative artificial intelligence for software engineering—a research agenda. *arXiv:2310.18648* (2023)
18. Ouyang, L., Wu, J., Jiang, X., Almeida, D., Wainwright, C., Mishkin, P., et al.: Training language models to follow instructions with human feedback. *arXiv:2203.02155* (2022)
19. Radford, A., Narasimhan, K., Salimans, T., Sutskever, I.: Language models are unsupervised multitask learners, *openAI Blog*, 2019
20. Raffel, C., Shazeer, N., Roberts, A., Lee, K., Narang, S., Matena, M., Zhou, Y., Li, W., Liu, P.J.: Exploring the limits of transfer learning with a unified text-to-text transformer. In: *Proceedings of the 36th International Conference on Machine Learning* (2019), *arXiv:1910.10683*
21. Tian, Y., Zhao, J., Dong, H., Xiong, J., Xia, S., Zhou, M., Lin, Y., Cambronero, J., He, Y., Han, S., et al.: Spreadsheetllm: Encoding spreadsheets for large language models. *arXiv:2407.09025* (2024)
22. Wang, C., Yang, Y., Gao, C., Peng, Y., Zhang, H., Lyu, M.R.: Prompt tuning in code intelligence: An experimental evaluation. *IEEE Transactions on Software Engineering* (11) (2023)
23. Wu, Z., et al.: Scaling laws for neural language models. *arXiv:2103.00020* (2021)
24. Zhu, Y.: Interactive ai systems for natural language understanding. *ACM Transactions on Interactive Intelligent Systems* **11**(3) (2021)