"This Sounds Unclear": Evaluating ChatGPT Capability in Translating End-User Prompts into Ready-to-Deploy Python Code

Margherita Andrao
Università di Trento
Fondazione Bruno Kessler
Trento, Italy

Diego Morra
Politecnico di Milano
Milan, Italy

Teresa Paccosi
Università di Trento
Fondazione Bruno Kessler
Trento, Italy

Maristella Matera
Politecnico di Milano
Milan, Italy

Barbara Treccani
Università di Trento
Trento, Italy

Massimo Zancanaro
Università di Trento
Fondazione Bruno Kessler
Trento, Italy

ABSTRACT

In this paper, we present a study aimed at evaluating how ChatGPT-4 understands end-users' natural language instructions to express automation rules for smart home applications and how it translates them into Python code ready to be deployed. Our study used 34 natural language instructions written by end users who were asked to automate scenarios presented as visual animations. The results show that ChatGPT-4 can produce coherent and effective code even if the instructions present ambiguities or unclear elements, understanding natural language instructions and autonomously resolving 94% of them. However, the generated code still contains numerous ambiguities that could potentially affect safety and security aspects. Nevertheless, when appropriately prompted, ChatGPT-4 can subsequently identify those ambiguities. This prompts a discussion about prospective interaction paradigms that may significantly improve the immediate usability of the generated code.

CCS CONCEPTS

• **Human-centered computing** → **Human computer interaction (HCI)**; HCI design and evaluation methods; User studies;

KEYWORDS

End-user development (EUD), Large language models (LLMs), ChatGPT-4, Task-automation systems

ACM Reference Format:
Margherita Andrao, Diego Morra, Teresa Paccosi, Maristella Matera, Barbara Treccani, and Massimo Zancanaro. 2024. "This Sounds Unclear": Evaluating ChatGPT Capability in Translating End-User Prompts into Ready-to-Deploy Python Code. In *International Conference on Advanced Visual Interfaces 2024 (AVI 2024), June 03–07, 2024, Arenzano, Genoa, Italy*. ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3656650.3656693

AVI 2024, June 03–07, 2024, Arenzano, Genoa, Italy

ACM ISBN 979-8-4007-1764-2/24/06
https://doi.org/10.1145/3656650.3656693

1 INTRODUCTION

AI-assisted code generator capabilities by Large Language Models (LLMs) are paving the way for new possibilities in the future of software development. While platforms such as Stack Overflow have previously offered entry-level dedicated support to those with at least some programming knowledge, the capability of models such as ChatGPT-4, CoPilot, and other specialized LLMs to generate code from natural language prompts is becoming increasingly pervasive among both beginner and expert programmers [11]. However, the existing literature still assumes that prompts are produced by experts in software development who can clearly and unambiguously articulate the requirements. Nevertheless, End-User Development (EUD), especially in the field of home automation, is one of the domains where this advancement has the potential to establish new standards. While the availability of user-friendly interfaces for commercial microcontrollers and sensors for the domotic Internet of Things (IoT) increases yearly, specific programming skills are still required to orchestrate and customize their operations. Research in EUD has proposed several approaches to facilitate naive users in defining those operations themselves, even without the need to acquire technical skills. Using trigger-action rules has proven an effective approach [7]. Enabling the creation of ready-to-deploy trigger-action rules from user-generated unconstrained natural language (NL) may significantly democratize the creation of home automation systems. However, there is still a need to explore the capability of LLMs to interpret instructions from naive users and generate ready-to-use code. The ability to interpret incorrect or ambiguous prompts that may not adhere to the structured format typical of trigger-action rules is crucial. This is especially important considering potential applications that aim to bridge non-expert user needs with rule-based systems.

This paper contributes to the ongoing discussion by presenting a study that explores how ChatGPT-4 understands ambiguous requests provided by end users and how it identifies and corrects ambiguities in the generated code.

2 RELATED WORKS

EUD investigates how naive users and non-professional developers can be enabled to create, modify, or extend software systems using a range of methods, techniques, and tools [3, 8]. Cloud-based

platforms, such as IFTTT[1], assist users in the definition of trigger-action rules for task-automation systems [5]. Research has explored the effectiveness of composition paradigms, including innovative visual paradigms and conversational-based approaches, broadening the scope of user interaction with smart-home technologies [2, 6, 10]. Recent works have turned attention to the influence of LLMs in this domain, emphasizing that interacting with ChatGPT can be challenging for end users who are not expert programmers due to NL ambiguities that hinder code generation [10, 13].

The quality and reliability of code generated by LLMs are increasing [9, 14, 15]. Some works highlighted ChatGPT's outperforming ability in generating and solving code problems in comparison to other models [1, 12]. Less has been done to assess how the interpretation of user prompts affects the output of reliable and functional code ready for deployment [13]. Further investigation is required to explore the reliability, cleanliness, and security of the code generated from inadequate or incomplete prompts. Especially in the EUD of IoT systems, where non-expert users are tasked with programming systems, security is crucial and makes these considerations particularly relevant. [4].

3 THE STUDY

Our study aimed to explore ChatGPT-4[2] capability to accurately generate correct code from trigger-action rules for home automation described by users using NL. Additionally, we aimed to examine how ChatGPT-4 assists users in recognizing ambiguities and detecting potential errors to create clearer, more accurate, and safer trigger-action rules.

Participants: Sixteen (16) participants, eight females and eight males, aged between 24 and 60 (M = 32.81; SD = 11.44) were involved in the study. Two had no prior experience with programming languages or home automation tools; six had minimal experience with smart home environments but no programming experience. The remaining eight (M = 4; F = 4) were expert programmers. Five of them had experience with smart home environments. All participants were native Italian speakers, and the instructions were produced in Italian.

Methods and procedure. Participants were exposed to 12 scenarios of smart home automation presented as silent video (to avoid language bias), and they were asked to write the rules to implement that automation. Each video lasted around 15 seconds and represented a combination of one state (e.g., "it's daytime"), one event (e.g., "the temperature rises"), and one action (e.g., "open the windows"). The initial segment of each video portrayed the smart home's response when the state is false (e.g., it's night, the temperature rises, and no action occurs), followed by the representation when the state is true. Each session was individual, and each participant had to write the instructions in natural language to explain to an "intelligent system" how to implement that automation.

Data analysis. In total, 199 instructions were collected (some participants wrote multiple independent/alternative rules to define the same scenario). Two researchers independently coded the 199 instructions as: (i) *non-ambiguous* if the instruction provided clear references to conditions and actions or *ambiguous* if the instruction contained unclear structure, terminology, or details that cast doubt on the outcome due to subjective interpretation and as (ii) *complete* if the instruction included all the elements (state, event, action) presented in the scenario, or *incomplete* if one or more elements were left implicit.

Only the 34 instructions evaluated as *ambiguous* and *complete* by both researchers were considered for further analyses. For each rule, we provided ChatGPT-4 with the instructions, the list of sensors and smart devices involved in the scenario (the context) and "questioned" it asking to: (i) Generate Python code ready to deploy (the prompt was: *"Given the following rule and the following context, make a Python code ready to be deployed. Just output the code without any further comments."*); (ii) Identify errors and ambiguities in the rule written by the user (the prompt was: *"Given the following rule and the context, identify if there are possible ambiguities or errors in the way the instructions were written."*).

Results. The 34 Python code snippets generated by ChatGPT-4 were analyzed by two independent researchers to evaluate the correctness, resulting in an accuracy rate of 94%. Only two of the generated codes were deemed incorrect. The first one lacked a condition explicitly stated in the prompt, while the second one overlooked an adverb that establishes the need for a time interval for the rule to be correctly executed. When prompted to identify ambiguities in the 34 natural language instructions, ChatGPT-4 detected 237 ambiguities in total, from a minimum of 3 to a maximum of 10 for each instruction (M = 6.97, SD = 1.45). Two researchers independently coded the descriptions of ambiguities and then examined if and how ChatGPT-4 had autonomously solved these ambiguities in the respective ready-to-deploy Python code snippet. Four main themes emerged from the ambiguities' descriptions.

Theme 1: Ambiguities related to the outcome of the rule and its variables (83 instances). Eleven (11) ambiguities involved uncertainties regarding the end of the action (*"The rule specifies 'after 7 PM' but does not indicate until when this rule applies."*). Nine (9) involved ambiguities related to temporal aspects as the rule did not clearly state the sequence/simultaneity of occurrences to trigger the action and two (2) involved suggestions for additional sensors or devices that, if integrated, can ensure the needed outcome. Other 39 involved possible ambiguities in error handling (*"The rule does not account for what should happen if the door fails to open or close."*), and specifically, in handling possible or imaginary conflicts, false negatives or positive triggers, or possible safety hazards. Finally, 22 described possible undesirable outcomes (e.g., automatically turning on a fireplace based on temperature and motion) associated with potentially harmful outcomes for safety, security risks, damages, and high/unusual energy consumption. Looking at the Python code snippets, 13 of these ambiguities were fully resolved by ChatGPT-4, four (4) were partially solved, and 66 were not addressed in the generated code.

Theme 2: Ambiguities related to the language (55 instances). Twenty-one (21) ambiguities involved word usage in formulating instructions (*"The rule mentions activating an evacuation plan but does not provide details on what the plan entails."*), with some words being overly specific, others being too generic or unconventional. Twenty-four (24) were related to the use of generic expressions, employing words instead of defining specific values (*"The rule does not specify what constitutes 'day.' Is it based on specific hours?"*) In

[1] https://ifttt.com/
[2] the latest available version at the study time in November 2023

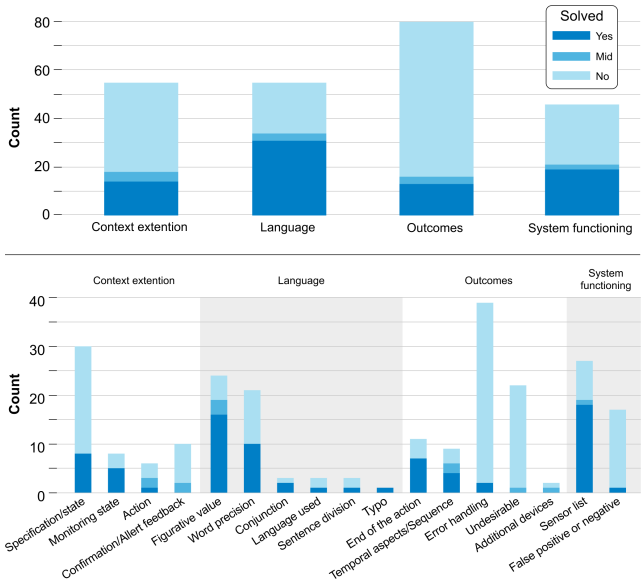

Figure 1: Distribution of solved ambiguities by theme (above) and identification code (below).

only one (1) instance, the ambiguity was related to an ambiguous typo in a rule. In three (3) cases, the ambiguity was related to the use of conjunctions, particularly the expression "and/or". In three (3), the ambiguity was related to a rule composed of multiple sentences while, in other three (3), it was related to the language used in general (*"If the smart home system's programming interface is in English or another language, the rule should be translated and formatted according to the system's requirements."*). In this group, 31 ambiguities were fully resolved in the code, three (3) were only partially resolved, and 21 had not been addressed.

Theme 3: Ambiguities related to the necessity of content extension (55 instances). This included: 30 cases of suggestions on specifying alternative states/conditions to avoid ambiguities; nine (9) cases of determining when/how often to monitor the state; six (6) cases for providing details of the action (*"The rule does not specify how much the curtains should close."*), and 10 cases concerning the need of confirmation/alert feedback (*"The rule does not include any provisions for alerting the occupants of the home that the window will be closed. This could be important for awareness and safety."*). Of these ambiguities, 14 were fully resolved in the generated code, four (4) partially, while 37 were not addressed.

Theme 4: Ambiguities related to the system's functioning (44 instances). This related to: 27 suggestions/comments on sensor localization and integration within a broader system; 17 accuracy/sensitivities of sensors that can lead to false positives or negatives triggers (*"It mentions a movement sensor, but movement sensors can sometimes give false positives or false negatives."*). Of these instances, 19 ambiguities were fully resolved, one (1) was partially resolved, and 24 were not addressed.

Overall, ChatGPT-4 was able to fully resolve 28% of the ambiguities above and to partially resolve 5% of them, while 67% were not addressed (see Fig 1). The analysis reveals ChatGPT-4's proficiency in resolving ambiguities related to the second theme *language*, successfully addressing 31 out of 55 cases. This is frequently attributed to its skill in interpreting incorrect verbal aspects or unconventional temporal and grammatical structures. However, ChatGPT-4 shows less expertise in dealing with ambiguities associated with the first theme *outcome*, as it replicated the same ambiguities found in the prompt in approximately 66 out of 80 cases. Notably, ChatGPT-4 tends to overlook two specific categories: error handling and ambiguities that might lead to unintended effects. Additionally, there were instances in each theme where ChatGPT-4 only partially resolved ambiguities. For example, in a case involving feedback to house occupants, the code included code debugging-level warning messages but not direct messages to the occupants. Another case involved the figurative interpretation of values, like defining "darkness" in a sentence. ChatGPT-4 identified the ambiguity but inadequately addressed it in the generated code. The code incorporated a logic to read data from the light sensor as a trigger, but it did not give the user a structure to set a threshold for the sensor data, offering only boolean true/false options for action triggering.

4 DISCUSSION AND CONCLUSION

The results provide evidence of the potential of LLMs, such as ChatGPT-4, in understanding NL instructions and translating them into functional code. Despite the ambiguities in the formulation, as well as the simplicity of the prompts and the context provided, ChatGPT-4 was able to consistently generate syntactically accurate and complete Python code. Furthermore, ChatGPT-4 could also detect many ambiguities in the users' expressions. Nevertheless, it fails, if not directly prompted, to properly recognize 67% of the ambiguities. Although these fails do not directly impact the generated code's correctness, they can bring to undesired effects or security issues. In our scenario, these aspects are even worse since our users would not be able to control the Python code and spot the issues. From our results, we can argue that an effective interaction for novice users should not rely on a direct code generation but it should involve ChatGPT-4 in initially identifying ambiguities, followed by iterative resolution processes that employ negotiation to ensure the accurate generation of code. We acknowledge some limitations in our study, starting from the limited number of participants that can affect the generalizability of the emerged themes. Additionally, ChatGPT-4 ability to address ambiguities could be mitigated by future model releases, emphasizing the importance of continuing research in this area. Nevertheless, we believe that this first study on automatically generated code from end users' natural language requests may help in future research such as on the design of effective interaction paradigms that facilitate the negotiation process to mitigate the ambiguities. This will require an extended research-through-design approach that addresses the multiple aspects emerging from the study outlined in this paper.

ACKNOWLEDGMENTS

This research received partial support from the PNRR project FAIR-Future AI Research (PE00000013), under the NRRP MUR program funded by the NextGenerationEU.

REFERENCES

[1] Imtiaz Ahmed, Ayon Roy, Mashrafi Kajol, Uzma Hasan, Partha Protim Datta, and Md Rokonuzzaman Reza. 2023. ChatGPT vs. Bard: a comparative study. *Authorea Preprints* (2023).

[2] Margherita Andrao, Fabrizio Balducci, Bernardo Breve, Federica Cena, Giuseppe Desolda, Vincenzo Deufemia, Cristina Gena, Maristella Matera, Andrea Mattioli, Fabio Paternò, et al. 2023. Understanding Concepts, Methods and Tools for End-User Control of Automations in Ecosystems of Smart Objects and Services. In *International Symposium on End User Development*. Springer, 104–124.

[3] Carmelo Ardito, Maria F. Costabile, Giuseppe Desolda, Marco Manca, Maristella Matera, Fabio Paternò, and Carmen Santoro. 2019. Improving Tools that Allow End Users to Configure Smart Environments. In *End-User Development*, Alessio Malizia, Stefano Valtolina, Anders Morch, Alan Serrano, and Andrew Stratton (Eds.). Springer International Publishing, Cham, 244–248.

[4] Bernardo Breve, Giuseppe Desolda, Francesco Greco, and Vincenzo Deufemia. 2023. Democratizing Cybersecurity in Smart Environments: Investigating the Mental Models of Novices and Experts. In *International Symposium on End User Development*. Springer, 145–161.

[5] Miguel Coronado and Carlos A. Iglesias. 2016. Task Automation Services: Automation for the Masses. *IEEE Internet Computing* 20, 1 (2016), 52–58. https://doi.org/10.1109/MIC.2015.73

[6] Giuseppe Desolda, Carmelo Ardito, and Maristella Matera. 2017. Empowering end users to customize their smart environments: model, composition paradigms, and domain-specific tools. *ACM Transactions on Computer-Human Interaction (TOCHI)* 24, 2 (2017), 1–52.

[7] Giuseppe Ghiani, Marco Manca, Fabio Paternò, and Carmen Santoro. 2017. Personalization of context-dependent applications through trigger-action rules. *ACM Transactions on Computer-Human Interaction (TOCHI)* 24, 2 (2017), 1–33.

[8] Henry Lieberman, Fabio Paternò, Markus Klann, and Volker Wulf. 2006. *End-User Development: An Emerging Paradigm*. Springer Netherlands, Dordrecht, 1–8. https://doi.org/10.1007/1-4020-5386-X_1

[9] Yue Liu, Thanh Le-Cong, Ratnadira Widyasari, Chakkrit Tantithamthavorn, Li Li, Xuan-Bach D. Le, and David Lo. 2023. Refining ChatGPT-Generated Code: Characterizing and Mitigating Code Quality Issues. arXiv:2307.12596 [cs.SE]

[10] Alberto Monge Roffarello and Luigi De Russis. 2023. Defining Trigger-Action Rules via Voice: A Novel Approach for End-User Development in the IoT. In *International Symposium on End User Development*. Springer, 65–83.

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

[12] Stephen R. Piccolo, Paul Denny, Andrew Luxton-Reilly, Samuel H. Payne, and Perry G. Ridge. 2023. Evaluating a large language model's ability to solve programming exercises from an introductory bioinformatics course. *PLOS Computational Biology* 19, 9 (09 2023), 1–16. https://doi.org/10.1371/journal.pcbi.1011511

[13] Gian Luca Scoccia. 2023. Exploring Early Adopters' Perceptions of ChatGPT as a Code Generation Tool. In *2023 38th IEEE/ACM International Conference on Automated Software Engineering Workshops (ASEW)*. 88–93. https://doi.org/10.1109/ASEW60602.2023.00016

[14] Burak Yetiştiren, Işık Özsoy, Miray Ayerdem, and Eray Tüzün. 2023. Evaluating the Code Quality of AI-Assisted Code Generation Tools: An Empirical Study on GitHub Copilot, Amazon CodeWhisperer, and ChatGPT. arXiv:2304.10778 [cs.SE]

[15] Li Zhong and Zilong Wang. 2023. Can ChatGPT replace StackOverflow? A Study on Robustness and Reliability of Large Language Model Code Generation. arXiv:2308.10335 [cs.CL]