2025 IEEE/ACM 47th International Conference on Software Engineering (ICSE)

Enhancing Code Generation via Bidirectional Comment-Level Mutual Grounding

Yifeng Di, Tianyi Zhang
Purdue University, West Lafayette, USA
di5@purdue.edu, tianyi@purdue.edu

***Abstract*—Large Language Models (LLMs) have demonstrated unprecedented capability in code generation. However, LLM-generated code is still plagued with a wide range of functional errors, especially for complex programming tasks that LLMs have not seen before. Recent studies have shown that developers often struggle with inspecting and fixing incorrect code generated by LLMs, diminishing their productivity and trust in LLM-based code generation. Inspired by the mutual grounding theory in communication, we propose an interactive approach that leverages code comments as a medium for developers and LLMs to establish a shared understanding. Our approach facilitates iterative grounding by interleaving code generation, inline comment generation, and contextualized user feedback through editable comments to align generated code with developer intent. We evaluated our approach on two popular benchmarks and demonstrated that our approach significantly improved multiple state-of-the-art LLMs, e.g., 17.1% pass@1 improvement for code-davinci-002 on HumanEval. Furthermore, we conducted a user study with 12 participants in comparison to two baselines: (1) interacting with GitHub Copilot, and (2) interacting with a multi-step code generation paradigm called Multi-Turn Program Synthesis. Participants completed the given programming tasks 16.7% faster and with 10.5% improvement in task success rate when using our approach. Both results show that interactively refining code comments enables the collaborative establishment of mutual grounding, leading to more accurate code generation and higher developer confidence.**

***Index Terms*—LLM, Code Generation, Code Refinement**

I. Introduction

The quest for automated code generation, dating back to the 1960s [1], [2], has evolved significantly. This field has transitioned from early deductive program synthesis methods [3]–[6] to the recent advent of Large Language Models (LLMs) [7]–[9]. Despite the significant progress, LLMs often fail to align with developer intent due to factors such as reliance on spurious features, lack of user context, and misunderstanding of complex specifications [10]–[14]. Recent efforts to address these limitations include model fine-tuning [15], [16], new prompting strategies [17], [18], and iterative refinement paradigms [19], [20]. However, the improvement brought by these methods is still limited. For example, Self-Debug [19] only achieved a 4.8% pass@1 increase for Codex on MBPP when test execution feedback is not available.

Recent studies [11], [21], [22] highlight the critical role of bi-directional communication between developers and LLMs in programming tasks. While developers can edit prompts, LLMs often treat such edits as new prompts, hindering their ability to understand and incorporate developers' refinement intent. Conversational models like ChatGPT offer multi-turn dialogues for feedback. However, effectively encoding historical utterances and contextualizing feedback within conversations remains a complex and challenging task [23]–[26].

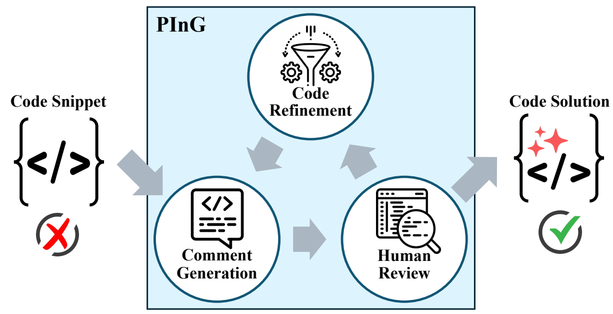


Fig. 1. Generating a code solution in our pipeline.

In this work, we propose a new interactive approach called Programming with Interactive Grounding (PING). Figure 1 illustrates this approach. PING employs inline comments as a medium for bi-directional communication between a developer and the model. This approach is inspired by the grounding theory in communication [27], which underscores the importance of mutual understanding in collaborative interactions. Unlike prior work that relies on coarse-grained code explanations for refinement [19], [28], PING's use of inline comments offers a more fine-grained approach. These comments directly address individual statements of the code, making it easier to target specific code segments for precise feedback and refinement.

Given a code snippet generated by an LLM, PING uses a code comment generation model to create inline comments that clarify each statement's behavior. These comments provide developers with an immediate, understandable code description, helping them quickly spot potential errors. Developers can edit the comments to specify the correct behavior for the erroneous statement. A code refinement model then regenerates the statement and subsequent segment identified by the feedback, rather than the entire code snippet.

Inline comments are essentially natural language descriptions, which are suitable for developers at all levels. This approach allows for more precise error identification compared to prompt editing or conversational models, which often require the model to infer the location of errors throughout the entire code snippet. Comment editing also directly indicates

[1]

[1] https://github.com/NecoraNyaru/PInG

Context c

```
def solve(N):
"""Given a positive integer N, return the total
sum of its binary digits in binary.
Example For N = 1000, the sum of digits will be
6 the output should be "110". For N = 150, the
sum of digits will be 4 the output should be
"100". For N = 147, the sum of digits will be 4
the output should be "100".
Variables: @N integer Constraints: 0 ≤ N ≤
10000.
Output: a string of binary number
"""
```

Code Snippet x

```
sum_of_digits = 0

digits = str(N)
for digit in digits:
    sum_of_digits += int(digit)

return bin(sum_of_digits)[2:]
```

1. Code Snippet with Comment $\dot{x}$

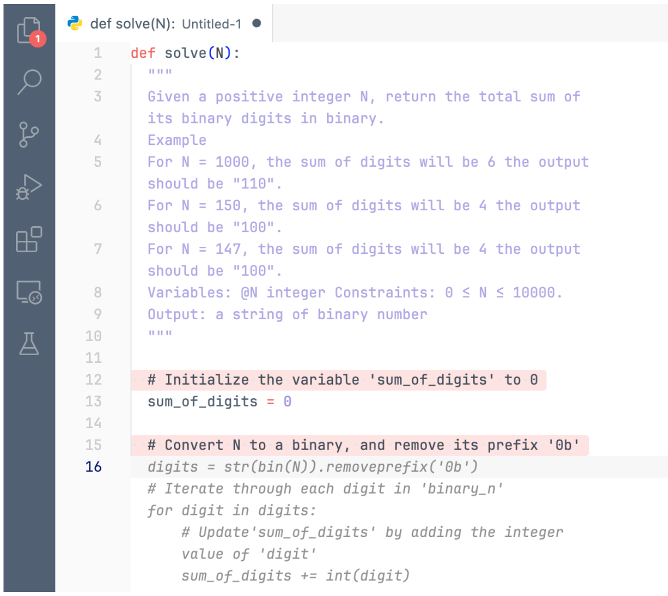
```
# Initialize the variable 'sum_of_digits' to 0
sum_of_digits = 0

# Convert N to a string
digits = str(N)
# Iterate through each digit in 'digits'
for digit in digits :
    # Update'sum_of_digits' by adding the integer
value of 'digit'
    sum_of_digits += int(digit)

# Convert the sum of digits to a binary, remove the
first two characters and return
return bin(sum_of_digits)[2:]
```

2. Refine the comment of the incorrect line of code

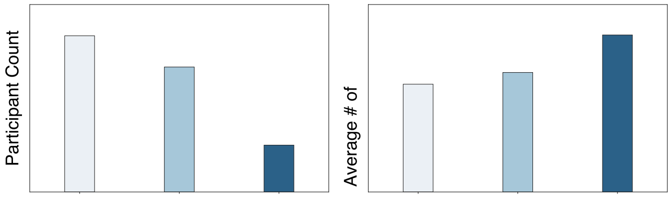
```
...
- # Convert N to a string
+ # Convert N to a binary, and remove its prefix '0b'
digits = str(N)
...
```

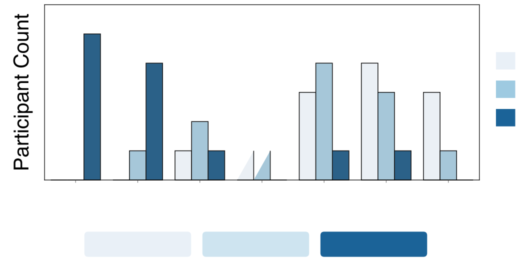
3. Regenerate based on the refined comment

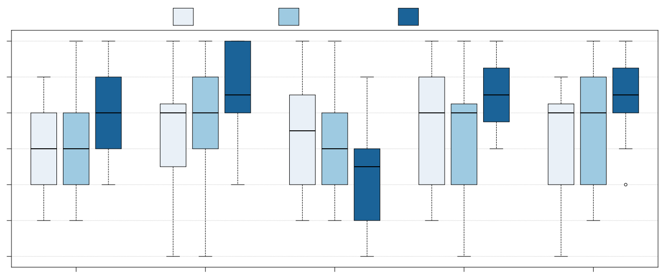
```
# Initialize the variable 'sum_of_digits'
to 0
sum_of_digits = 0

# Convert N to a binary and remove its
prefix '0b'
- digits = str(N)
+ digits = str(bin(N)).removeprefix('0b')
# Iterate through each digit in
'binary_n'
for digit in digits:
    # Update'sum_of_digits' by adding the
integer value of 'digit'
    sum_of_digits += int(digit)

# Convert the sum of digits to a binary,
remove the first two characters and
return
return bin(sum_of_digits)[2:]
```


def solve(N): Untitled-1
def solve(N):
"""
Given a positive integer N, return the total sum of its binary digits in binary.
Example
For N = 1000, the sum of digits will be 6 the output should be "110".
For N = 150, the sum of digits will be 4 the output should be "100".
For N = 147, the sum of digits will be 4 the output should be "100".
Variables: @N integer Constraints: 0 ≤ N ≤ 10000.
Output: a string of binary number
"""
# Initialize the variable 'sum_of_digits' to 0
sum_of_digits = 0
# Convert N to a binary, and remove its prefix '0b'
digits = str(bin(N)).removeprefix('0b')
# Iterate through each digit in 'binary_n'
for digit in digits:
# Update'sum_of_digits' by adding the integer value of 'digit'
sum_of_digits += int(digit)

(1)


Participant Count
Average # of


Participant Count



REFERENCES

[1] R. J. Waldinger and R. C. Lee, "Prow: A step toward automatic program writing," in *Proceedings of the 1st international joint conference on Artificial intelligence*, 1969, pp. 241–252.

[2] P. D. Summers, "A methodology for lisp program construction from examples," *Journal of the ACM (JACM)*, vol. 24, no. 1, pp. 161–175, 1977.

[3] R. M. Burstall and J. Darlington, "A transformation system for developing recursive programs," *Journal of the ACM (JACM)*, vol. 24, no. 1, pp. 44–67, 1977.

[4] Z. Manna and R. Waldinger, "Synthesis: dreams$\rightarrow$ programs," *IEEE Transactions on Software Engineering*, no. 4, pp. 294–328, 1979.

[5] ——, "A deductive approach to program synthesis," *ACM Transactions on Programming Languages and Systems (TOPLAS)*, vol. 2, no. 1, pp. 90–121, 1980.

[6] D. R. Smith, "Top-down synthesis of divide-and-conquer algorithms," *Artificial Intelligence*, vol. 27, no. 1, pp. 43–96, 1985.

[7] M. Chen, J. Tworek, H. Jun, Q. Yuan, H. P. d. O. Pinto, J. Kaplan, H. Edwards, Y. Burda, N. Joseph, G. Brockman *et al.*, "Evaluating large language models trained on code," *arXiv preprint arXiv:2107.03374*, 2021.

[8] Y. Li, D. Choi, J. Chung, N. Kushman, J. Schrittwieser, R. Leblond, T. Eccles, J. Keeling, F. Gimeno, A. Dal Lago *et al.*, "Competition-level code generation with alphacode," *Science*, vol. 378, no. 6624, pp. 1092–1097, 2022.

[9] R. Li, Y. Zi, N. Muennighoff, D. Kocetkov, C. Mou, M. Marone, C. Akiki, L. Jia, J. Chim, Q. Liu *et al.*, "Starcoder: may the source be with you!" *Transactions on Machine Learning Research*, 2023.

[10] B. Kou, S. Chen, Z. Wang, L. Ma, and T. Zhang, "Do large language models pay similar attention like human programmers when generating code?" *Proceedings of the ACM on Software Engineering*, vol. 1, no. FSE, pp. 2261–2284, 2024.

[11] P. Vaithilingam, T. Zhang, and E. L. Glassman, "Expectation vs. experience: Evaluating the usability of code generation tools powered by large language models," in *Chi conference on human factors in computing systems extended abstracts*, 2022, pp. 1–7.

[12] H. Yu, B. Shen, D. Ran, J. Zhang, Q. Zhang, Y. Ma, G. Liang, Y. Li, Q. Wang, and T. Xie, "Codereval: A benchmark of pragmatic code generation with generative pre-trained models," in *Proceedings of the 46th IEEE/ACM International Conference on Software Engineering*, 2024, pp. 1–12.

[13] Z. Z. Wang, A. Asai, X. V. Yu, F. F. Xu, Y. Xie, G. Neubig, and D. Fried, "Coderag-bench: Can retrieval augment code generation?" *arXiv preprint arXiv:2406.14497*, 2024.

[14] Z. Wang, Z. Zhou, D. Song, Y. Huang, S. Chen, L. Ma, and T. Zhang, "Where do large language models fail when generating code?" *arXiv preprint arXiv:2406.08731*, 2024.

[15] P. Haluptzok, M. Bowers, and A. T. Kalai, "Language models can teach themselves to program better," *arXiv preprint arXiv:2207.14502*, 2022.

[16] A. Chen, J. Scheurer, J. A. Campos, T. Korbak, J. S. Chan, S. R. Bowman, K. Cho, and E. Perez, "Learning from natural language feedback," *Transactions on Machine Learning Research*, 2024. [Online]. Available: https://openreview.net/forum?id=xo3hI5MwvU

[17] C. Wang, Y. Yang, C. Gao, Y. Peng, H. Zhang, and M. R. Lyu, "No more fine-tuning? an experimental evaluation of prompt tuning in code intelligence," in *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2022, pp. 382–394.

[18] Y. Choi and J.-H. Lee, "Codeprompt: Task-agnostic prefix tuning for program and language generation," in *Findings of the Association for Computational Linguistics: ACL 2023*, 2023, pp. 5282–5297.

[19] X. Chen, M. Lin, N. Schärli, and D. Zhou, "Teaching large language models to self-debug," in *The Twelfth International Conference on Learning Representations*, 2024. [Online]. Available: https://openreview.net/forum?id=KuPixIqPiq

[20] Y. Dong, X. Jiang, Z. Jin, and G. Li, "Self-collaboration code generation via chatgpt," *arXiv preprint arXiv:2304.07590*, 2023.

[21] C. Bird, D. Ford, T. Zimmermann, N. Forsgren, E. Kalliamvakou, T. Lowdermilk, and I. Gazit, "Taking flight with copilot: Early insights and opportunities of ai-powered pair-programming tools," *Queue*, vol. 20, no. 6, pp. 35–57, 2022.

[22] S. Barke, M. B. James, and N. Polikarpova, "Grounded copilot: How programmers interact with code-generating models," *Proceedings of the ACM on Programming Languages*, vol. 7, no. OOPSLA1, pp. 85–111, 2023.

[23] C.-W. Liu, R. Lowe, I. V. Serban, M. Noseworthy, L. Charlin, and J. Pineau, "How not to evaluate your dialogue system: An empirical study of unsupervised evaluation metrics for dialogue response generation," *arXiv preprint arXiv:1603.08023*, 2016.

[24] Z. Tian, R. Yan, L. Mou, Y. Song, Y. Feng, and D. Zhao, "How to make context more useful? an empirical study on context-aware neural conversational models," in *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, 2017, pp. 231–236.

[25] C. Sankar, S. Subramanian, C. Pal, S. Chandar, and Y. Bengio, "Do neural dialog systems use the conversation history effectively? an empirical study," *arXiv preprint arXiv:1906.01603*, 2019.

[26] W. Zheng and K. Zhou, "Enhancing conversational dialogue models with grounded knowledge," in *Proceedings of the 28th ACM International Conference on Information and Knowledge Management*, 2019, pp. 709–718.

[27] L. B. Resnick, J. M. Levine, and S. D. Teasley, *Perspectives on socially shared cognition.* American Psychological Association, 1991.

[28] X. Jiang, Y. Dong, L. Wang, Q. Shang, and G. Li, "Self-planning code generation with large language model," *arXiv preprint arXiv:2303.06689*, 2023.

[29] S. Yao, J. Zhao, D. Yu, N. Du, I. Shafran, K. R. Narasimhan, and Y. Cao, "React: Synergizing reasoning and acting in language models," in *The Eleventh International Conference on Learning Representations*, 2022.

[30] S. Yao, D. Yu, J. Zhao, I. Shafran, T. L. Griffiths, Y. Cao, and K. Narasimhan, "Tree of thoughts: Deliberate problem solving with large language models," *arXiv preprint arXiv:2305.10601*, 2023.

[31] W. Wang, Y. Wang, S. Joty, and S. C. Hoi, "Rap-gen: Retrieval-augmented patch generation with codet5 for automatic program repair," in *Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2023, pp. 146–158.

[32] K. Zhang, Z. Li, J. Li, G. Li, and Z. Jin, "Self-edit: Fault-aware code editor for code generation," *arXiv preprint arXiv:2305.04087*, 2023.

[33] J. Li, G. Li, Y. Li, and Z. Jin, "Enabling programming thinking in large language models toward code generation," *arXiv preprint arXiv:2305.06599*, 2023.

[34] H. Le, H. Chen, A. Saha, A. Gokul, D. Sahoo, and S. Joty, "Codechain: Towards modular code generation through chain of self-revisions with representative sub-modules," *arXiv preprint arXiv:2310.08992*, 2023.

[35] "GitHub Copilot · Your AI pair programmer," 2023. [Online]. Available: https://github.com/features/copilot

[36] E. Nijkamp, B. Pang, H. Hayashi, L. Tu, H. Wang, Y. Zhou, S. Savarese, and C. Xiong, "Codegen: An open large language model for code with multi-turn program synthesis," *arXiv preprint arXiv:2203.13474*, 2022.

[37] H. H. Clark and E. F. Schaefer, "Contributing to discourse," *Cognitive science*, vol. 13, no. 2, pp. 259–294, 1989.

[38] H. H. Clark and G. L. Murphy, "Audience design in meaning and reference," in *Advances in psychology*. Elsevier, 1982, vol. 9, pp. 287–299.

[39] S. E. Brennan and H. H. Clark, "Conceptual pacts and lexical choice in conversation." *Journal of experimental psychology: Learning, memory, and cognition*, vol. 22, no. 6, p. 1482, 1996.

[40] E. A. Isaacs and H. H. Clark, "Ostensible invitations1," *Language in society*, vol. 19, no. 4, pp. 493–509, 1990.

[41] Y. Zhang, S. Sun, M. Galley, Y.-C. Chen, C. Brockett, X. Gao, J. Gao, J. Liu, and B. Dolan, "Dialogpt: Large-scale generative pre-training for conversational response generation," *arXiv preprint arXiv:1911.00536*, 2019.

[42] K. R. Chandu, Y. Bisk, and A. W. Black, "Grounding'grounding'in nlp," *arXiv preprint arXiv:2106.02192*, 2021.

[43] Z. Feng, D. Guo, D. Tang, N. Duan, X. Feng, M. Gong, L. Shou, B. Qin, T. Liu, D. Jiang *et al.*, "Codebert: A pre-trained model for programming and natural languages," *arXiv preprint arXiv:2002.08155*, 2020.

[44] S. Lu, D. Guo, S. Ren, J. Huang, A. Svyatkovskiy, A. Blanco, C. Clement, D. Drain, D. Jiang, D. Tang *et al.*, "Codexglue: A machine learning benchmark dataset for code understanding and generation," *arXiv preprint arXiv:2102.04664*, 2021.

[45] H. Husain, H.-H. Wu, T. Gazit, M. Allamanis, and M. Brockschmidt, "Codesearchnet challenge: Evaluating the state of semantic code search," *arXiv preprint arXiv:1909.09436*, 2019.

[46] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[47] D. Guo, Q. Zhu, D. Yang, Z. Xie, K. Dong, W. Zhang, G. Chen, X. Bi, Y. Wu, Y. Li *et al.*, "Deepseek-coder: When the large language model meets programming–the rise of code intelligence," *arXiv preprint arXiv:2401.14196*, 2024.

[48] D. Kocetkov, R. Li, L. Ben Allal, J. Li, C. Mou, C. Muñoz Ferrandis, Y. Jernite, M. Mitchell, S. Hughes, T. Wolf, D. Bahdanau, L. von Werra, and H. de Vries, "The stack: 3 tb of permissively licensed source code," *Preprint*, 2022.

[49] R. Rubinstein, "The cross-entropy method for combinatorial and continuous optimization," *Methodology and computing in applied probability*, vol. 1, pp. 127–190, 1999.

[50] J. Austin, A. Odena, M. Nye, M. Bosma, H. Michalewski, D. Dohan, E. Jiang, C. Cai, M. Terry, Q. Le *et al.*, "Program synthesis with large language models," *arXiv preprint arXiv:2108.07732*, 2021.

[51] Y. Liu, C. Tantithamthavorn, Y. Liu, and L. Li, "On the reliability and explainability of automated code generation approaches," *arXiv preprint arXiv:2302.09587*, 2023.

[52] D. Fried, A. Aghajanyan, J. Lin, S. Wang, E. Wallace, F. Shi, R. Zhong, W.-t. Yih, L. Zettlemoyer, and M. Lewis, "Incoder: A generative model for code infilling and synthesis," *arXiv preprint arXiv:2204.05999*, 2022.

[53] M. L. McHugh, "Interrater reliability: the kappa statistic," *Biochemia medica*, vol. 22, no. 3, pp. 276–282, 2012.

[54] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," *Advances in neural information processing systems*, vol. 27, 2014.

[55] Y. Huang, H. Guo, X. Ding, J. Shu, X. Chen, X. Luo, Z. Zheng, and X. Zhou, "A comparative study on method comment and inline comment," *ACM Transactions on Software Engineering and Methodology*, vol. 32, no. 5, pp. 1–26, 2023.

[56] F. F. Xu, B. Vasilescu, and G. Neubig, "In-ide code generation from natural language: Promise and challenges. corr, abs/2101.11149 (2021)," *arXiv preprint arXiv:2101.11149*, 2021.

[57] Y. Lai, C. Li, Y. Wang, T. Zhang, R. Zhong, L. Zettlemoyer, W.-t. Yih, D. Fried, S. Wang, and T. Yu, "Ds-1000: A natural and reliable benchmark for data science code generation," in *International Conference on Machine Learning*. PMLR, 2023, pp. 18 319–18 345.

[58] S. G. Hart and L. E. Staveland, "Development of nasa-tlx (task load index): Results of empirical and theoretical research," in *Advances in psychology*. Elsevier, 1988, vol. 52, pp. 139–183.

[59] R. C. Martin, *Clean code: a handbook of agile software craftsmanship*. Pearson Education, 2009.

[60] Z. Luo, C. Xu, P. Zhao, Q. Sun, X. Geng, W. Hu, C. Tao, J. Ma, Q. Lin, and D. Jiang, "Wizardcoder: Empowering code large language models with evol-instruct," in *The Twelfth International Conference on Learning Representations*, 2023.

[61] Y. Wei, Z. Wang, J. Liu, Y. Ding, and L. Zhang, "Magicoder: Source code is all you need," *arXiv preprint arXiv:2312.02120*, 2023.

[62] M. Bavarian, H. Jun, N. Tezak, J. Schulman, C. McLeavey, J. Tworek, and M. Chen, "Efficient training of language models to fill in the middle," *arXiv preprint arXiv:2207.14255*, 2022.