Extending ClarifyCoder to Real-World Developer Needs

1 Introduction

ClarifyCoder was proposed to address a fundamental challenge in code generation: developers often issue underspecified or ambiguous requests. Instead of producing flawed code, ClarifyCoder first issues a clarifying question and only then generates a solution when requirements are complete [1]. This clarify-then-code paradigm improves model alignment in controlled benchmarks, but there remains a gap between research prototypes and real-world developer workflows.

This paper explores how ClarifyCoder can be extended into real environments—particularly IDEs, GitHub issue triaging, and Computing Education—and what guardrails are necessary to ensure productive adoption.

2 Challenges in Real-World Adoption

Developers operate inside editors (VS Code, JetBrains) and collaboration platforms (GitHub Issues/PRs). A model must fit seamlessly into these workflows rather than forcing a separate interaction channel [2]. Another concern is hallucination. Code LLMs frequently generate incorrect APIs or packages, which can break builds or introduce security risks [3, 4]. Finally, user trust is central: while copilots increase productivity, empirical studies stress the importance of human-in-the-loop review [5]. In educational settings, over-reliance on generated answers may inhibit learning if clarifications are not structured properly [6].

3 Proposed Extensions

3.1 IDE Integration

ClarifyCoder can be embedded into IDEs to intercept ambiguity at the source. Ambiguous TODOs or comments can be flagged with an "Ambiguity" marker. When ambiguity is detected, the model asks a single clarifying question in a side panel, and developer answers are stored in a ledger for transparency. Once clarified, ClarifyCoder proposes a patch as a unified diff, requiring explicit developer approval. This approach aligns with evidence that dialog-style refinement improves code generation outcomes [5].

3.2 GitHub Issue Triage and PR Review

In GitHub Issues, ClarifyCoder can act as a triage bot. If critical details (e.g., environment, reproduction steps) are missing, it posts one clarifying comment and labels the issue as `needs-info`. If sufficient, it generates acceptance criteria. For pull requests, ClarifyCoder can read diffs and CI logs; if unclear, it asks a clarifying question, otherwise it proposes small actionable review items. Automated review systems show measurable productivity benefits when carefully scoped and kept under human oversight [2].

3.3 Computing Education

In student mode, ClarifyCoder can scaffold specification skills by asking stepwise clarifications (e.g., "What range?" and "What format?") before returning a solution. This reinforces precise problem definition rather than bypassing it. In instructor mode, educators can define which parts of a problem are clarifiable. The Q/A ledger can be reviewed during grading for transparency. Studies in education suggest generative AI can support formative assessment, but alignment with rubrics requires explicit grounding [6].

4 Guardrails and Safety

To mitigate risks, ClarifyCoder should:

- Require diffs rather than raw code insertions, running patches through linters or unit tests [4].
- Enforce a one-question cadence to reduce workflow interruptions [1].
- Apply dependency allowlists to prevent package hallucinations [3].
- Store Q/A ledgers for auditability in pull requests and learning management systems.

5 Evaluation Metrics

Effectiveness can be measured by:

- **Developer productivity:** time-to-first-commit, percent of issues resolved after one ClarifyCoder question, reduction in PR review latency.
- **Code quality:** regression rates in ClarifyCoder-touched lines, dependency violation frequency.
- **Educational impact:** rubric alignment, pass rates with ≤ 2 clarifications, time-on-task reductions.

6 Conclusion

ClarifyCoder's clarify-then-code behavior addresses a critical gap in current coding assistants: handling ambiguity explicitly. By embedding this capability into IDEs, GitHub workflows, and educational platforms—with guardrails such as diff-first proposals, one-question loops, and ledger logging—ClarifyCoder can evolve from a research prototype into a practical productivity and learning tool. This will reduce costly mis-specifications in professional development while also training students in the art of precise problem definition.

References

[1] Wu, Jie, et al. *ClarifyCoder: Clarification-Aware Code LLMs*. arXiv preprint arXiv:2504.16331, 2025. https://arxiv.org/abs/2504.16331

[2] Ziegler, Daniel, et al. *Challenges in Integrating LLMs with Developer Workflows*. arXiv preprint arXiv:2310.12005, 2023. https://arxiv.org/abs/2310.12005

[3] Wang, Xia, et al. *Package Hallucinations in Code LLMs*. arXiv preprint arXiv:2312.02127, 2023. https://arxiv.org/abs/2312.02127

[4] Sobania, Dominik, et al. *Hallucination Patterns in Code LLMs*. arXiv preprint arXiv:2403.03163, 2024. https://arxiv.org/abs/2403.03163

[5] Bird, Christian, et al. *Field Study of AI Coding Assistants in Practice*. arXiv preprint arXiv:2401.02028, 2024. https://arxiv.org/abs/2401.02028

[6] Khosravi, H., et al. *Generative AI in Education: Opportunities and Risks*. Computers & Education: Artificial Intelligence, 2023. https://doi.org/10.1016/j.caeai.2023.100177