2026 ASEE ANNUAL CONFERENCE June 21–24, 2026 | Charlotte Convention Center, NC
Host Sponsor UNC Charlotte
ASEE AMERICAN SOCIETY FOR ENGINEERING EDUCATION
Paper ID #52337

LLM conversation transcripts: redefining homework in the age of generative AI

Dr. Edward F. Gehringer, North Carolina State University at Raleigh

Dr. Gehringer is a professor in the Departments of Computer Science, and Electrical & Computer Engineering. His research interests include data mining to improve software-engineering practice, and improving assessment through machine learning and natural language processing.

Prerak Manish Bhandari, North Carolina State University at Raleigh

LLM Conversation Transcripts in Software Engineering: Redefining Homework in the Age of Generative AI

Abstract

Students are already using large language models in their coursework, making the central question for engineering educators not whether to respond but how. This paper presents an observational analysis of LLM use in an upper-level software engineering course and argues that unguided access tends to support efficient implementation more than the development of design reasoning. By analyzing complete student–LLM interaction transcripts from two substantial projects, the study shows that students engage actively with AI tools but rarely use them to explore alternatives, interrogate trade-offs, or articulate rationale. These findings suggest that LLM use must be taught explicitly: instructors should model productive prompting strategies, align assessment with design-centered objectives, and incorporate interaction transcripts as reflective artifacts. Framing LLMs as structured learning tools rather than unregulated aids offers a pragmatic path for integrating AI into engineering education while preserving the integrity of design-focused learning goals.

1. Introduction: Teaching Programming in the Age of Large Language Models

Large language models (LLMs) have rapidly become part of everyday programming practice. Students now routinely consult tools such as ChatGPT to debug errors, generate code snippets, summarize documentation, and propose architectural solutions. In educational settings, the use of LLMs is no longer an edge case but a default behavior. As a result, the central question facing computing educators is no longer whether students will use generative AI, but how that use should be shaped to support learning rather than undermine it.

This paper adopts a deliberately pedagogical framing of the problem: *Students need to learn how to use LLMs in programming, but how do we teach them to do so productively?* In introductory programming courses, especially those focused on syntax or narrowly scoped problems, LLM assistance may appear largely benign. However, in advanced courses, such as those focused on object-oriented design (OOD), the risks and opportunities are more pronounced. OOD emphasizes abstraction, responsibility assignment, separation of concerns, and the justification of trade-offs. These competencies are difficult to assess through functional correctness alone and can be weakened when students rely on AI systems without engaging in design-level reasoning.

NCSU CSC/ECE 517, Object-Oriented Design and Development, provides a compelling context in which to study this challenge. The course emphasizes architectural reasoning, maintainability, and principled design through substantial, team-based software projects. In such settings, LLMs can act as powerful assistants by providing rapid feedback and reducing friction in implementation. At the same time, they can obscure gaps in understanding by producing plausible, working solutions that students may not fully comprehend or be able to defend. A correct implementation, therefore, does not necessarily indicate mastery of design.

This study examines how students interact with LLMs during the design and implementation process. This study does not introduce a new LLM-based system or instructional intervention; rather, it focuses on analyzing naturally occurring student–LLM interactions to understand how these tools are used in practice. We analyze transcripts of student–LLM conversations submitted as part of two major course projects. These transcripts provide direct evidence of student problem-solving behavior, including the types of questions students ask, how they iterate on solutions, and whether they seek explanations or direct fixes. This process-oriented perspective allows us to move beyond outcome-based evaluation toward a more nuanced understanding of AI-assisted learning.

The motivation for this approach is twofold. First, discussions of AI in coursework often focus on detection or prohibition, neither of which reflects the reality that AI-assisted development is increasingly standard in professional practice. Second, prior research suggests that the way students seek help—whether they request explanations and rationale or simply ask for solutions—has meaningful implications for learning. Understanding these patterns in a design-focused course is essential for developing evidence-based instructional guidance.

This paper presents an empirical study of LLM use in courses, grounded in systematic analysis of student–LLM interactions across two team-based projects. We characterize help-seeking behavior along multiple dimensions, including the balance between executive and instrumental prompts, the granularity of requests, and the presence of explicit design reasoning. Our goal is not to evaluate individual students, but to identify recurring interaction patterns that have pedagogical significance.

The contributions of this paper are threefold. First, we present an observational, process-level analysis of how students in an upper-level object-oriented design course interact with LLMs during authentic project work. Second, we provide a structured characterization of these interactions using dimensions such as help-seeking type, prompt granularity, and presence of design reasoning, grounded in real student–LLM transcripts. Third, we interpret these interaction patterns in the context of design-oriented learning objectives and derive pedagogical implications for integrating LLMs into software engineering education.

The remainder of the paper situates this study within related work, describes the course context and data collection process, presents empirical results, and discusses how object-oriented design education can adapt to an AI-augmented learning environment.

2. Background and Motivation

The integration of large language models into programming education introduces challenges and opportunities that vary substantially by course level and learning objectives. In courses centered on object-oriented design (OOD), these issues are particularly acute. This section explains why OOD courses represent a critical setting for studying LLM use and why examining student interaction processes is necessary to understand learning outcomes.

2.1 Object-Oriented Design as a Learning Objective

Object-oriented design courses differ fundamentally from introductory programming courses. While early courses often emphasize syntax, control flow, and basic data structures, OOD courses prioritize

abstraction, responsibility assignment, modularity, and long-term maintainability. Students are expected to justify design decisions using established object-oriented principles such as cohesion, coupling, and responsibility-driven design in particular ways, how responsibilities are allocated, and how designs support extension and change.

Assessment in such courses is therefore inherently challenging. Functional correctness is a necessary but insufficient condition for success. Two implementations may exhibit identical behavior while reflecting very different levels of design quality and conceptual understanding. For this reason, OOD pedagogy traditionally relies on design artifacts, written justification, code reviews, and iterative refinement to surface student reasoning.

The availability of LLMs complicates this landscape. Modern models can generate code that conforms to common design heuristics and framework conventions without explicitly articulating the rationale behind those choices. When students adopt such solutions without fully understanding them, the resulting code may appear well designed while masking gaps in conceptual mastery. This tension makes OOD courses a particularly important domain for studying how students interact with and rely on generative AI.

2.2 Generative AI as a Learning Actor

LLMs occupy an ambiguous role in programming education. They can function as tutors by explaining concepts and answering questions, as debugging assistants that help diagnose errors, and as code generators that produce complete implementations with minimal prompting. Each of these roles has distinct implications for learning.

In professional software development, these tools are increasingly accepted as productivity aids. Students are aware of this reality and often view LLM use as practical and expected. Educational settings, however, impose goals beyond productivity, including the development of transferable reasoning skills and the ability to explain and defend design decisions.

Without explicit guidance, students may default to interaction patterns that prioritize short-term results over conceptual learning. This tendency is reinforced in project-based courses with substantial implementation demands, where time pressure can encourage solution-seeking behavior. As a result, the educational impact of LLMs depends on how students engage with them.

2.3 Why Studying Process Matters More Than Outcomes

Much of the existing discourse around AI use in coursework focuses on final artifacts, such as submitted code, which provide limited insight into how students arrive at their solutions. Analyzing student–LLM conversations offers a different perspective. These interactions reveal what students choose to ask, whether they seek explanations or direct fixes, how they respond to errors, and whether they revisit earlier decisions. They also expose the extent to which students articulate design intent versus delegating reasoning to the model. By situating LLM use within the specific learning objectives of an object-oriented design course and examining interaction processes rather than only final outcomes, this work seeks to demonstrate how generative AI is currently being used and how it might be taught more effectively.

3. Related Work

Research on large language models in computing education has expanded rapidly, spanning programming assistance, embedded chatbots, AI literacy, and assignment design [4–7]. This section reviews work most relevant to the present study and highlights gaps motivating an empirical analysis of LLM use in an advanced object-oriented design course.

3.1 LLMs in Programming and Engineering Education

Studies of LLM use in programming courses consistently find that targeted assistance—such as concept clarification or debugging—can support learning, whereas relying on LLMs for full solutions is less effective [4, 5]. Case studies show that many students either avoid LLMs or use them selectively for specific subtasks rather than outsourcing entire assignments [4].

Research on mandatory or embedded LLM use reports mixed outcomes: overall performance often resembles prior offerings, but students vary widely in how they engage with the tool [5, 8]. Students who treat the LLM as an answer engine tend to perform worse, while those who iterate and refine responses engage more productively. These studies also note increased assessment overhead and the need for clearer guidance on effective interaction [5, 8].

Overall, the educational impact of LLMs depends more on instructional framing and assignment structure than on tool availability alone [5, 7, 8].

3.2 Help-Seeking Behavior and Learning Outcomes

A substantial body of work examines help-seeking behavior, distinguishing executive help-seeking (requesting solutions) from instrumental help-seeking (seeking explanations or partial guidance). Frequent executive help-seeking correlates with weaker outcomes, whereas instrumental help-seeking aligns with deeper understanding [1, 2].

Recent studies extend this framework to LLM interactions by classifying prompts by intent, granularity, and novelty. Solution-oriented queries are associated with shallow engagement, while explanation-oriented prompts better support learning [1, 3]. Automated classification methods have also been shown feasible for large-scale analysis of developer–LLM interactions [1].

However, most research focuses on introductory or intermediate courses. Little is known about help-seeking patterns in advanced, design-focused settings, where the boundary between implementation help and design outsourcing is more consequential [2, 3].

3.3 Assignment Design and Assessment in the Presence of LLMs

Another research strand examines how assignments and assessments can be adapted to LLM capabilities. Some work emphasizes designing tasks resistant to full automation—for example, by adding visual components, multi-file complexity, or tightly constrained requirements [7]. Evaluations show that while LLMs often produce correct code, they frequently rely on advanced features or violate constraints, revealing use of AI-generated solutions [7].

Other studies argue for shifting assessments toward explanation, justification, and reflection [5, 8]. In design-focused courses, this includes articulating design intent, justifying architectural choices, and explaining trade-offs. LLM-generated explanations often describe what code does rather than why it exists, underscoring the need to foreground design rationale [2, 8]. Together, this work highlights the need for pedagogical strategies that shape how students engage with LLMs rather than simply allowing or prohibiting their use [5, 7, 8].

3.4 Gaps in Prior Work

Despite growing interest in generative AI, several gaps remain. Few studies examine LLM use in upper-level, team-based software engineering courses where design reasoning is central [2, 3]. Many analyses rely on final submissions or self-reports rather than detailed interaction records [4, 5]. Empirical work directly linking LLM interaction patterns to object-oriented design learning objectives is also limited [2].

The present study addresses these gaps by analyzing student-submitted LLM transcripts from two major projects. By examining how students interact with LLMs during authentic design and implementation tasks, it contributes process-level evidence that complements outcome-focused research and informs more nuanced instructional guidance.

4. Course Context and Study Design

This study is situated within CSC/ECE 517, Object-Oriented Design and Development, a beginning masters-level course that emphasizes principled software design, architectural reasoning, and long-term maintainability.

The course places explicit emphasis on object-oriented design principles such as responsibility assignment, separation of concerns, low coupling, high cohesion, and extensibility. Students are expected not only to implement working features but also to justify their design decisions, explain trade-offs between alternative approaches, and demonstrate how their designs support future modification. Assessment includes evaluation of design artifacts (such as class structure and component interactions), written or verbal justification of design choices, and the ability to reason about how systems evolve under changing requirements. As a result, design reasoning is a central learning objective, and correctness alone is not sufficient to demonstrate mastery.

The data for this study comes from two major team-based projects completed by pairs or trios of students in a class of about thirty. The projects required students to design, implement, and extend nontrivial features in a shared codebase, reasoning about class responsibilities, component interactions, and the design implications for extensibility and maintainability.

Students were required to use large language models during their work and submit the transcripts of their interactions. The course treated these tools as part of contemporary practice and examined their impact directly. This requirement promoted transparency, reduced incentives for undisclosed outsourcing, and gave instructors insight into how students engaged with AI tools. For the study, it also provided detailed process data capturing students' problem-solving behavior over time.

Students were reminded that they remained responsible for understanding and justifying their design and implementation choices, regardless of AI assistance. The transcripts were treated as supplementary artifacts rather than graded content, limiting incentives to curate or polish interactions.

The study employs an observational design, analyzing naturally occurring student–LLM interactions rather than directing or constraining how students used the tools. This approach captures authentic usage patterns, including both productive and unproductive behaviors.

Situated in an advanced object-oriented design course and drawing on complete transcripts from two major projects, the study provides an empirically grounded view of contemporary LLM use in upper-level software design. This context informs the data collection and analysis that follow.

This emphasis on design reasoning makes the course a particularly relevant setting for examining whether LLM interactions support or bypass the development of object-oriented design skills.

5. Data Collection and Corpus

This section describes the data used in the study, how it was collected, and why it is appropriate for examining student interaction with large language models in an object-oriented design course. The emphasis is on capturing authentic student behavior during AI-assisted project work rather than inferred usage or post hoc self-reports.

Students submitted links to their LLM interaction sessions. There were nine projects in each of the two rounds. Each transcript captures a multi-turn interaction between a student team member and an LLM, typically spanning task planning, debugging, implementation requests, and refinement cycles. The length and depth of interactions vary, but together these transcripts provide a detailed record of how students engaged with LLMs while working on authentic design tasks.

Conversations were included if they met the following criteria:

- The interaction was directly related to project work, including design discussion, implementation, debugging, or testing.
- The transcript captured a meaningful exchange rather than a single isolated prompt.
- The content was sufficiently complete to support classification along the analytical dimensions used in the study.

Conversations that were empty, duplicated, or unrelated to course work were excluded. When multiple transcripts were submitted by a single team, each transcript was treated as an independent interaction instance, representing distinct problem-solving episodes rather than aggregated team behavior.

The use of student LLM interaction data raises ethical considerations related to consent, privacy, and academic integrity. Students were informed that their transcripts could be used for research purposes in an anonymous form, and participation did not affect grading. The transcripts were preserved in their original textual form. No attempts were made to normalize language, correct grammar, or restructure prompts.

However, all transcripts were anonymized before analysis to remove identifying information, including student names, team identifiers, repository URLs, and references to private credentials.

Participation in transcript submission was required as part of the course, while any accompanying survey responses (if applicable) were voluntary. Out of 27 students, 23 submitted survey responses (~85%). As a result, survey participation may not represent the entire class population and is subject to self-selection bias. This study does not rely on survey data for statistical inference; instead, its primary analysis is based on submitted LLM interaction transcripts. The findings should therefore be interpreted as descriptive of observed interaction patterns rather than as statistically generalizable results.

6. Analysis Method and Results

This section describes the analytical approach used to examine student–LLM interactions and presents the resulting quantitative summaries. The goal is to characterize observed interaction patterns in an object-oriented design course, not to generalize beyond this instructional context.

The unit of analysis in this study is an individual student prompt extracted from submitted LLM interaction transcripts. A total of 70 prompts were analyzed across the two projects.

6.1 Illustrative examples of prompt types

To clarify the coding scheme, we provide representative examples of student prompts from the transcript corpus. All examples are anonymized and lightly edited for brevity.

Example 1: Executive, low-level prompt

“Fix this Ruby method so that it passes the failing test cases.”

This prompt directly requests a solution without asking for explanation or rationale.

Example 2: Instrumental prompt with design focus

“Why would using a Factory pattern be better than directly instantiating objects in this controller?”

This prompt seeks explanation and comparison, indicating engagement with design reasoning.

Example 3: Design language without exploration

“Use a Singleton pattern here and show the implementation.”

Although design terminology is used, the prompt remains solution-oriented and does not explore alternatives or justification.

Example 4: Iterative debugging prompt

“I updated the method as suggested but now I get this error. What should I change?”

This reflects iterative interaction focused on resolving runtime issues rather than revisiting design decisions.

Each prompt was coded along three dimensions:

1. **Help-seeking type.** Prompts were classified as either executive or instrumental. Executive prompts request direct fixes, generated artifacts, or actionable commands. Instrumental prompts request explanations, rationale, or discussion of trade-offs.

2. **Prompt granularity.** Prompts were categorized as low, mid, or high granularity. Low-level prompts address method-level code, configuration, or test failures. Mid-level prompts concern component placement or interactions between classes. High-level prompts address architecture, design patterns, or structural trade-offs.

3. **Explicit design reasoning presence.** A prompt was marked as containing design reasoning if it explicitly asked why an approach was appropriate, requested comparison of alternatives, or referenced design principles or patterns. The presence of design reasoning is treated as a necessary but not sufficient indicator of design engagement.

These examples illustrate that while design-related terminology appears in some prompts, it is often embedded within solution-seeking interactions rather than used to explore alternatives.

Coding was performed conservatively using rule-based criteria to minimize over-interpretation.

Table 1 summarizes the distribution of help-seeking types by project. Executive prompts constitute the majority of student–LLM interactions in both projects. While Project 2 shows a modest increase in instrumental prompts, executive help-seeking remains dominant overall. Figure 1 shows the same information graphically.

Table 1. Help-seeking type by project

	Total prompts	Executive (#, %)		Instrumental (#, %)	
1st Project	16	13	81%	3	19%
2nd Project	54	41	76%	13	24%
Combined	70	54	77%	16	23%

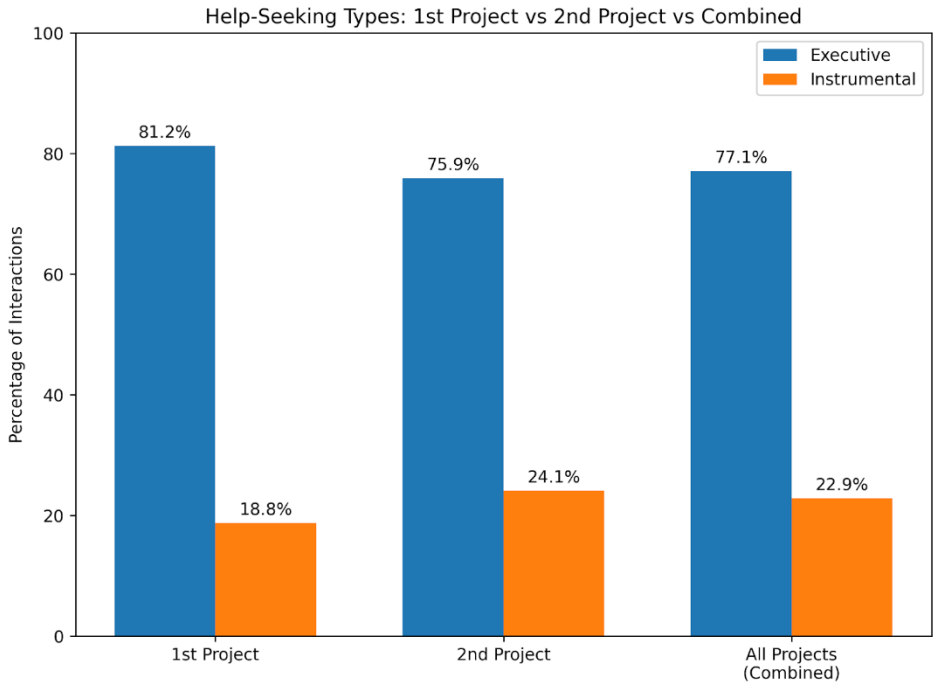

Figure 1. Distribution of help-seeking types (all projects).

Table 2 summarizes prompt granularity across the two projects. Low-level prompts related to debugging and configuration are common in both projects. High-level prompts appear in both projects but do not constitute a majority in Project 2 despite increased project complexity.

Table 2. Prompt granularity by project

	Total prompts	Low (#, %)		Mid (#, %)		High (#, %)	
1st Project	16	5	31%	3	19%	8	50%
2nd Project	54	22	41%	11	20%	21	39%

Table 3 reports the presence of explicit design reasoning indicators. Explicit design reasoning appears in a majority of prompts in both projects. However, the presence of design-oriented language alone does not imply sustained or student-initiated design exploration.

Table 3. Prompts with explicit design reasoning

	Total prompts	Design reasoning present (#, %)		absent (#, %)	
1st Project	16	11	69%	3	31%
2nd Project	54	39	72%	11	28%

In summary, student interaction with LLMs is characterized by a predominance of executive help-seeking, frequent low-level prompts, and intermittent use of design-oriented language. Differences

between the two projects are incremental rather than transformative, with increased project complexity associated with slightly more instrumental prompting.

7. Discussion: What Are Students Learning from LLM Use?

The interpretations in this section are derived directly from the interaction patterns reported in Section 6. Table 1 shows that a large majority of prompts (77%) are executive in nature, indicating that students most often request direct solutions or fixes rather than explanations. Table 2 further indicates that many of these prompts are low-level, focusing on debugging and implementation details rather than architectural or design concerns. Although Table 3 shows that design-related language appears in a majority of prompts, closer inspection reveals that such language is often embedded within solution-seeking queries rather than used to explore alternatives or justify decisions. Taken together, these results suggest that while students engage actively with LLMs, their interactions are primarily oriented toward completing tasks efficiently rather than developing or articulating design reasoning.

It is important to clarify the scope of these interpretations. This study does not directly measure student learning outcomes, such as conceptual understanding, retention, or the ability to independently explain or defend design decisions. Instead, it analyzes observable interaction patterns with LLMs as indicators of how students engage with problem-solving tasks. The conclusions presented here should therefore be understood as inferences about learning-related behavior rather than direct evidence of learning gains or deficits.

The interaction patterns observed in Section 6 suggest that students use LLMs effectively to complete projects, but that the learning process may be uneven and often misaligned with the goals of an object-oriented design course. The prevalence of executive help-seeking indicates that students treat LLMs primarily as tools for accelerating implementation and fixing errors rather than for developing design-level understanding.

Consequently, students appear to be learning how to operate within a codebase more than how to reason about its structure—an effect amplified by working in an unfamiliar language (Ruby). Low-level prompts reduce development friction but do not require articulating design intent or considering alternatives, allowing students to reach functional solutions without engaging deeply with responsibility allocation, architectural trade-offs, or maintainability.

Design-oriented language in prompts complicates interpretation but does not contradict it. Although students often use design terms, these terms typically accompany requests for direct answers. In many cases, the LLM—not the student—introduces design ideas, which students accept without probing or rephrasing, suggesting that exposure does not equate to internalized understanding.

Iteration patterns reinforce this conclusion. Students iterate in response to runtime failures, not design concerns, and rarely revisit earlier decisions once code works. This distinction between iterative coding and iterative design is especially important in an object-oriented context.

These findings do not indicate misuse. Students employ LLMs efficiently and in ways consistent with professional practice. The issue is one of alignment: without explicit guidance, LLM use supports short-term productivity more reliably than the development of design reasoning.

Overall, simply requiring LLM use in design-focused courses is insufficient. If instructors want students to reason about and defend design decisions, they must teach students how to engage with LLMs in ways that promote reflection, critique, and comparison of alternatives rather than direct solution adoption.

8. Pedagogical Implications

LLM use should be taught as a skill. Students are comfortable using LLMs for code generation and debugging but less practiced at using them to explore alternatives, weigh trade-offs, or articulate rationale. Instruction should model and reward prompts that seek explanation and comparison rather than direct solutions, using templates that foreground design intent and alternative allocations of responsibility.

Aligning assessment with design-centered learning objectives. Assessment strongly shapes how students use LLMs. When correctness dominates grading, students naturally optimize for working code. Emphasizing explanation, justification, and reflection encourages deeper engagement. Requiring students to defend decisions independently of LLM output, critique AI suggestions, or document unimplemented alternatives reduces reliance on answer-seeking and promotes critical interaction.

Using LLM transcripts as reflective artifacts. LLM transcripts can support metacognitive reflection rather than serve only as accountability records. Asking students to annotate where they accepted suggestions uncritically or where different prompts might have yielded better insight helps them recognize when they are outsourcing reasoning versus engaging in it.

Balancing authentic practice and educational goals. In industry, efficiency is paramount; in design courses, reasoning is. The goal is not to restrict LLM use but to structure it so that professional tools support learning. Without explicit guidance, LLMs tend to reinforce implementation-focused behavior; with deliberate instructional design, they can instead promote deeper engagement with object-oriented design principles.

9. Conclusion and Future Work

This paper examined how students in an advanced object-oriented design course used large language models during substantial, team-based programming projects. By analyzing student-submitted LLM interaction transcripts rather than relying solely on final artifacts or self-reported usage, the study provides a process-level view of how generative AI is incorporated into authentic design and implementation work.

Returning to the motivating question—*students need to learn how to use LLMs in programming, but how do we teach them?*—the findings show that such learning does not emerge on its own. When LLM use is allowed without guidance, students gravitate toward efficiency and error-fixing rather than reflective engagement with design principles. Although this behavior is reasonable and mirrors professional practice, it does not reliably advance the goals of an object-oriented design course.

At the same time, the results do not suggest misuse. Students engage actively with LLMs, and design-related concepts appear in their prompts. The challenge is therefore not to limit access but to align LLM use with instructional aims. Teaching students to ask more probing questions, critique AI suggestions, and articulate design rationale independently becomes a central pedagogical task.

This study contributes empirical evidence to ongoing discussions about generative AI in computing education by examining an upper-level, design-focused context and analyzing interaction processes rather than outcomes alone. It complements prior work on assignment design, help-seeking, and AI-assisted learning while underscoring the distinctive challenges of courses where design reasoning is a primary objective.

Several directions for future work remain. Longitudinal studies could examine how student interaction patterns with LLMs evolve across multiple courses or over the duration of a degree program. Intervention-based studies could evaluate the impact of explicit instruction in effective LLM use on design reasoning and learning outcomes. Scalable approaches to analyzing LLM interaction data could support instructors in providing targeted feedback and refining course design.

As generative AI becomes a permanent fixture of software development practice, the responsibility of computing educators is not to resist this change, but to shape it. By teaching students how to engage with LLMs thoughtfully and critically, object-oriented design education can continue to emphasize its core goal: developing engineers who can reason about software, not merely produce it.

A key limitation of this study is the absence of direct measures of student learning. Future work should investigate how LLM interaction patterns relate to learning outcomes, including students' ability to explain their code, justify design decisions, and perform independently in assessment settings without AI assistance. Comparative studies across cohorts with and without LLM access, as well as controlled interventions, would provide a more complete understanding of the educational impact of generative AI.

References

[1] S. Zhong, Y. Zou, and B. Adams, "Developer–LLM conversations: An empirical study of interactions and generated code quality," *arXiv preprint arXiv:2509.10402*, 2025.

[2] J. D. Zamfirescu-Pereira, E. Jun, M. Terry, and Q. Yang, "Beyond code generation: LLM-supported exploration of the program design space," in *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI)*, 2025.

[3] J. Ullrich, M. Koch, and A. Vogelsang, "From requirements to code: Understanding developer practices in LLM-assisted software engineering," in *Proceedings of the IEEE International Conference on Software Engineering (ICSE)*, 2025.

[4] A. Deriyeva, J. Dannath, and B. Paaßen, "Case study: Using LLMs to assist with solving programming homework assignments," in *Proceedings of the DELFI Workshops*, 2024.

[5] L. Elson, L. Gaughan, B. Hopton, and S. Lloyd-Brown, "'ChatGPT did my homework': Lessons learnt from embedding a chatbot in undergraduate coursework," *Physics Education*, 2025.

[6] T. van der Velden, "Using LLMs for support in high school programming education," Delft University of Technology Repository, 2024.

[7] B. McDanel and E. Novak, “Designing LLM-resistant programming assignments: Insights and strategies for CS educators,” in *Proceedings of the ACM Technical Symposium on Computer Science Education (SIGCSE)*, 2025.

[8] E. F. Gehringer, “In the age of LLMs, is dual-submission homework dead?” in *Proceedings of the ASEE Annual Conference and Exposition*, 2025.