An Empirical Study of Anonymous, Unmoderated, and Online Peer-to-Peer Programming Tutoring Conversations

Priscila Santiesteban
University of Michigan
Ann Arbor, Michigan, USA
pasanti@umich.edu

Emma Shedden
University of Pennsylvania
Philadelphia, Pennsylvania, USA
eshedden@upenn.edu

Madeline Endres
University of Massachusetts Amherst
Amherst, Massachusetts, USA
mendres@umass.edu

Westley Weimer
University of Michigan
Ann Arbor, Michigan, USA
weimerw@umich.edu

Abstract

Many people want to learn to program but lack access to traditional classroom instruction. Teaching these novices at scale is crucial for building a more diverse and capable software engineering workforce. While online tools like Stack Overflow and ChatGPT offer help, they can be impersonal or reinforce poor software development practices. Anonymous peer-to-peer (P2P) tutoring has the potential to be an additional place for scalable support, but we lack a firm understanding of how to best support it for CS pedagogy.

We present a mixed-analysis study of n=108 anonymous, unmoderated P2P CS tutoring sessions. We analyze text-based conversations from Python Tutor, a widely-used learning platform. In this setting, novice programmers (Learners) request help from volunteer programmers (Helpers) in a shared coding environment. We present a qualitatively-backed model of user motivations, conversational dynamics, and Learner-reported satisfaction. Surprisingly, Learners often receive useful (59% of tutoring interactions), low-toxicity (78% of messages) help without moderation. P2P chats reflect key phases of the software development process (83% of chats) and occasionally foster personal connection (17% of chats). We identify behaviors linked to satisfaction and discuss implications for scalable peer tutoring system design for CS education.

CCS Concepts

• **Applied computing** → **Collaborative learning**; • **Social and professional topics** → **Software engineering education**; *Informal education*; • **Information systems** → **Chat**.

ACM Reference Format:
Priscila Santiesteban, Emma Shedden, Madeline Endres, and Westley Weimer. 2026. An Empirical Study of Anonymous, Unmoderated, and Online Peer-to-Peer Programming Tutoring Conversations. In *2026 IEEE/ACM 48th International Conference on Software Engineering (ICSE-SEET '26), April 12–18, 2026, Rio de Janeiro, Brazil.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3786580.3786958

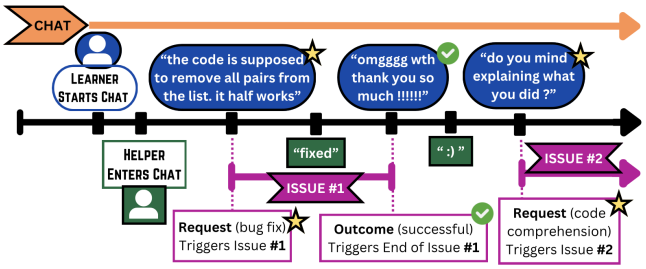

Figure 1: Visualization of a Python Tutor chat, showing how *issues* are segmented without overlap within a larger *chat* (5 messages shown out of 36). The Learner (blue circle) initiates; a Helper (green square) joins later. Each issue begins with a Learner 's *request* and ends with an *outcome* (e.g., expression of satisfaction by Learner). All intervening messages are annotated with "Communication" types.

1 Introduction

Students learning to program are increasingly turning to online resources for help, from MOOCs (massive open online courses) [44] and programming forums such as Stack Overflow [19, 52, 60] to newer options like Discord communities [4, 15, 59] and conversational AI tools such as ChatGPT [20, 83]. Universities are even beginning to structure and personalize such support through institutional Large Language Model (LLM) deployments (e.g., ASU [79], Harvard [55], etc.). This growing reliance on online resources may reflect several persistent challenges in computing education: enrollments [41, 42] that may strain available TA budgets, limited opportunities for non-traditional learners to access timely support [10], and uneven retention [49, 50, 68]. These challenges disproportionately affect students from underrepresented groups, who are more likely to feel unsupported [29].

Online resources offer partial solutions to these problems by expanding access to help while reducing reliance on in-person staff. While usable at scale, these platforms are not without limitations. Sites like StackOverflow or MOOCS can feel impersonal or

ICSE-SEET '26, Rio de Janeiro, Brazil

ACM ISBN 979-8-4007-2423-7/26/04
https://doi.org/10.1145/3786580.3786958

unmotivating [6, 14], leading to low retention in practice [45]. In contrast, AI tools often provide direct answers that may not foster deeper reflection or learning [63, 83]; AI over-reliance negatively impacts cognitive skills such as decision-making, critical thinking, and analytical reasoning [84], all of which are essential for learning software engineering [26]. Thus, students using online resources may feel isolated or receive help that resolves an immediate coding problem but does little to support long-term growth or persistence.

One potential solution is online *peer-to-peer tutoring* (P2P) [18, 87]. In this setup, programming novices (Learners) can request help from fellow novices or more experienced volunteers (Helpers), creating opportunities for personalized support that may better mirror traditional teaching environments. In anonymous and unmoderated contexts, it could be flexible and scalable, applicable to both traditional and non-traditional students. However, it is unclear if such settings lead to improved programming learning outcomes and satisfaction. While researchers have considered aspects of this problem in isolation (e.g., the benefits of social features in *formal* online learning [5], unmoderated P2P interactions *without* a learning component [22], or online education with *trained* tutors [40, 56]), we are not aware of research on the use of anonymous, unmoderated, and untrained P2P tutoring in computing education.

We help fill this gap through a mixed analysis study of $n = 108$ anonymous P2P tutoring sessions (comprising 40,755 total words). We analyze P2P sessions from Python Tutor, a popular online code editor for novices that provides step-by-step execution visualizations [37, 38]. Python Tutor piloted a P2P feature where Learners could join a help queue and be paired with anonymous volunteer Helpers, communicating through a text chat and shared live coding environment.[1] These P2P conversations were anonymous, unmoderated, and could involve arbitrary code, making them an excellent dataset for evaluating if P2P tutoring can effectively assist with programming help at scale in a supportive and inclusive manner.

We contribute the first qualitatively characterized model of anonymous, unmoderated P2P programming tutoring:

- **Who uses P2P tutoring and why.** We find that most Learners are CS1-level novices, seeking help with writing new code and debugging. Helpers range from peers to professional engineers.
- **What occurs in P2P tutoring sessions.** Conversations are overwhelmingly positive in tone and frequently extend beyond debugging to cover software development processes such as requirements, design, and testing.
- **How outcomes are shaped.** We identify conversational strategies linked to learner satisfaction, a key factor for persistence and retention in computing education [7].

2 Background and Terminology

With over 10 million users globally, ***Online Python Tutor*** is an educational tool for visualizing program execution, allowing users to write, run and step through arbitrary code in a sandbox environment [37, 38]. While aimed at novices and sometimes used by university programming students, Python Tutor attracts a diverse

[1]While the specific chat feature studied here is not currently available on Python Tutor, other large sites, such as Khan Academy's associated *schoolhouse.world*, also offer free online P2P tutoring for computer science where "no experience is required" to become a volunteer peer tutor [47].

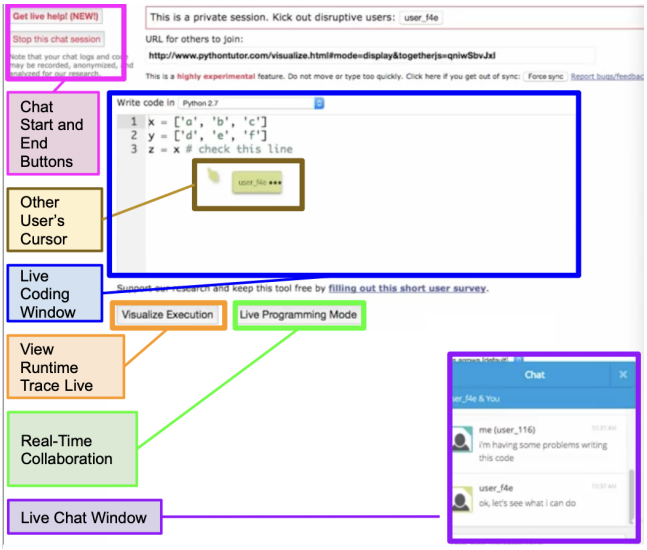

Figure 2: Learner view of Python Tutor.

range of users including self-taught learners, hobbyists, adult students, and educators exploring alternative learning methods [38, Fig. 1]. This broad user base contrasts with the more homogeneous groups typically found in traditional classroom settings.

While primarily a code execution platform, Python Tutor also piloted a real-time peer-to-peer chat feature for live coding communication [39]. A ***chat*** is a platform-supported session initiated by a user, and includes all natural language ***messages*** (i.e., a text input followed by pressing enter) and shared code edits in that session. Designed to mimic tutoring, chat sessions could be private or public (open to all users). This paper focuses on archival public chats.

Figure 2 shows the online interface as it appeared for the texts analyzed here. When a user clicked "Get Live Help", Python Tutor created a random chat ID and showed the session, along with the user's country, to other users who could choose to join and help. All sessions contained a live chat window in the bottom-right corner of the screen. Similar to instant messaging, users sent messages visible to everyone in the session. To share live code edits, users could activate "Live Programming Mode", which included a coding window for real-time collaboration. Any participant could run the program and observe its behavior ("Visualize Execution"), but only the session creator could end the chat session or kick other users out. In the rest of this paper, we refer to the user who starts a session (i.e., opens the chat) and requests help as the ***Learner***, and any volunteers who may join and exit at any point during the session and send a message as ***Helpers***. To validate the distinction between Learners and Helpers, we applied likelihood ratio testing to compare nested GLMER models with and without user type (Learner/Helper) as a factor (Section 4.3). We found that including user type significantly improves model fit ($p < 2.2e{-}16$), providing strong evidence that Learners and Helpers exhibit distinct overall patterns in chat conversations.

Initially, Python Tutor only supported private chats, requiring Learners to know a tutor or peer Helper and externally share a URL before the session [39]. A study on that private-only version found that it successfully connected geographically-separated

peers and facilitated learning in tutoring and collaborative learning sessions. However, that approach limited scalability and made the chats inaccessible to Learners who did not already know a peer or tutor. As a result, Python Tutor enabled public chats (the focus of this paper). Critically, in public chats *any number of anonymous, untrained Helpers could join or leave at any point during the session, and chats were unmoderated.* We present the first analysis of these anonymous P2P interactions, investigating how novice programmers actually carry out P2P tutoring in practice.

3 Chat Structure and Unit of Analysis

To ground our analytical approach, we first describe the conversational patterns that emerged from an exploratory review of a subset of Python Tutor chat transcripts (Section 4.2.1). These insights informed our chat segmentation and outcome measure choices.

Chats Contain Multiple Issues. Most chats begin with a Learner (defined in Section 2) presenting a programming-related request (e.g., debugging, writing new code; see Section 5.1.2). However, conversations often shift over time: after resolving an initial problem, Learners may introduce new questions or explore related concepts, even if different Helpers join or leave. To capture this recurring structure, we segmented chats into discrete *help-seeking episodes*, which we refer to as **issues** (Figure 1). Thus, our analysis considers both the full chat and its constituent issues as units of analysis. In total, we analyze 108 chats that contained 133 issues.

Outcomes Defined by Learner Satisfaction. We evaluated each issue based on Learner satisfaction, as expressed within the chat (e.g., explicit gratitude, positive closure, or frustration). While this measure does not directly capture conceptual learning, it reflects whether the immediate tutoring goal was met from the Learner 's perspective. Standard learning metrics (e.g., program compilation, test scores) are poorly suited to this context: code may compile while still failing to meet a learner's intent, and goals often evolve mid-conversation. Satisfaction is linked to motivation and retention in computing education [6, 7, 35].

We are particularly interested in self-reported satisfaction of social tutoring given our focus on understanding online pedagogy — including non-traditional novices. Salguero *et al.* found that "lack of sense of belonging" (which included feeling supported, etc.), a social factor, explained more of the variance in early CS outcomes than other factors such as in-class confusion, personal obligations or lack of confidence [68, Tab. 2]. Similarly, Lehman *et al.* found that "peer experiences in computing are central to student persistence in the major" [50]. Overall, in a meta-review of 50 publications, Badali *et al.* found that social motivations (perceptions of support, interaction, connecting with others, etc.) were the second most frequent factor for student retention in online classes [6, Tab. 2], after only the direct academic motive (e.g., earning credit). Other studies focus more directly on learning outcomes (e.g., course grades [49]); our research questions focus on self-perceptions of satisfaction with tutoring support, and our dataset permits a nuanced investigation.

4 Analysis Approach

We take a mixed analysis approach to assess anonymous, unmoderated peer-to-peer (P2P) programming tutoring chats on Python Tutor. Our analysis is guided by the following research questions:

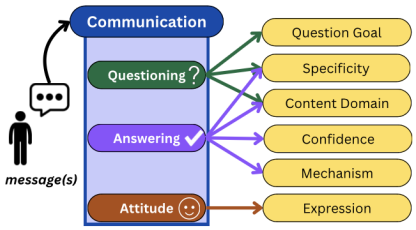

Figure 3: User messages' conversational function is identified via labeling with "Communication" codes. Each type has axis associated with it to capture further conversational depth. All types can be used by Learners and Helpers.

RQ1: *Why* do Learners use Python Tutor?
RQ2: *How* are Helpers characterized on Python Tutor?
RQ3: *What* is the content of Python Tutor interactions?
RQ4: *How* do people feel about their Python Tutor interactions?
RQ5: *What* leads to successful interactions on Python Tutor?

We apply qualitative content analysis (Section 4.2) to a random sample of Python Tutor chat transcripts (Section 4.1), developing a two-part coding scheme that segments chats into distinct issues, then labels the communicative function of users' messages (Section 4.2.2). We also use these codes as features in generalized linear mixed-effects regression models (Section 4.3) to go beyond description and explore predictive relationships (e.g., whether certain communication strategies are more likely to lead to positive outcomes).

4.1 Analysis Dataset

We collected initiation events from the Python Tutor platform [37] over six days between October and November 2019. Not every event reflected tutoring, so we applied automated filtering and manual review to construct a dataset of programming interactions.

We retained only non-trivial tutoring chats, requiring at least two participants (Learner and Helper) and more than five messages, yielding 1,314 sessions. Very long sessions (over 150 messages, 3%) were excluded, as annotation costs rise with transcript length and fatigue can reduce consistency in long qualitative sessions [16], leaving 1,158 sessions. From this pool, we randomly sampled chats and had three annotators verify whether a programming-related issue was present; sampled sessions were discarded if conversations started midstream, lacked code, or were not about programming, in which case another random session was drawn.

The final dataset consisted of 108 chats (108 Learners and 266 Helpers), comprising 8,973 messages and 40,755 words. While moderate in size, this dataset was large enough to capture recurring patterns in tutoring interactions (Section 4.2.2) and to support reliable coding across annotators (Section 4.2.4).

Our replication package contains our annotated anonymized dataset, scripts, and assumption checking. It is available at: https://github.com/pasantiesteban/ICSE_SEET2026_Chats.

4.2 Qualitative Analysis Methodology

We used *qualitative content analysis* [13, 85] to systematically examine the structure and content of Python Tutor chats. This approach allowed us to identify communication strategies, Learners' help-seeking goals, issues' outcomes, and the basic structure of tutoring

conversations. We developed a two-part coding scheme through iterative annotation, combining deductive insights (e.g., prior research and domain expertise) with inductive analysis of emergent patterns. The scheme includes codes that segment chats into distinct issues (see Section 3) and codes that label the communicative function of user messages. These were consolidated into a structured *codebook* and applied with high inter-rater reliability (Section 4.2.4). Beyond supporting descriptive insights, these codes also served as inputs to our statistical models (Section 4.3).

4.2.1 Method to Generate Codebook. An initial exploratory review of a small set of chats (excluded from the main analysis) revealed recurring patterns in user roles, programming experience, chat structure, and content. Based on these observations, prior work (Section 7), and our research questions, two researchers iteratively *open coded* [48] a separate sample of 100 chats. Through discussion, we refined the resulting codes into a structured codebook and coding scheme (Section 4.2.2). This process allowed us to iteratively refine the codebook until thematic saturation was reached—that is, no new codes or categories emerged. Once the codebook was stable, three annotators (including one of the initial coders) then applied the finalized scheme to a new set of 108 chats. The first 10 were coded collaboratively to reach a second round of code saturation with the new annotators. The remaining chats were annotated independently, with weekly meetings to resolve disagreements—seeking full consensus on the number and type of issues.

4.2.2 Codebook, Codes, and, Coding Scheme. Our codebook includes five top-level code families (set in Title Case): "Issue Request", "Issue Outcome", and three types of "Communication", collectively totaling 46 sub-codes (set in lower case). The first two support segmentation of chats into discrete issues (Section 3), while the latter three characterize message-level conversational behavior.

We apply two structural codes to segment each chat into issues. An "Issue Request" marks a Learner 's request for programming help, which we classify into four categories, i.e., sub-codes (see Section 5.1.2). This is followed by a corresponding "Issue Outcome" indicating that the Learner expressed either satisfaction (e.g., thanks) or dissatisfaction (e.g., confusion, disengagement). Each *request–outcome* pair defines a self-contained issue (see Section 3). We identify 133 issues within the 108 chats.

For all other messages, we apply "Communication" codes to characterize conversational behavior. At a high level, we identify three principal message types: "Questioning" (i.e., form of query for information), "Answering" (i.e., form of giving information), and "Attitude" (i.e., form of emotional expression). These can all be used by both Learners and Helpers. For example, a Learner might ask a question regarding some programming syntax, or a Helper might pose a question to guide the Learner through the problem-solving process. Further, each type has between one and four axes (see Figure 3). For example, "Questioning" messages are annotated with a "Question Goal" (e.g., to check correctness, gather information, etc.), "Content Domain" (the discussion topic, e.g., "bug", "error message", etc.), and "Specificity" ("vague"/"detailed"). See Table 1 for more. These sub-codes allow us to capture rich communication.

4.2.3 Applying Codes. We apply codes to one or more sequential chat *messages* (i.e., text input followed by pressing enter) from a user. We applied "Issue Request" and "Issue Outcome" structural codes only to Learners' messages, as only Learners initiate and resolve issues (see Section 3). For all other messages, code application were constrained to reflect a single "Communication" type: "Questioning", "Answering", or "Attitude". For example, if a Learner sends two sequential messages—*"I am worried"* and *"Can you help me with arrays?"*—each would be its own coding unit ("Attitude", then "Questioning"). If two or more sequential messages had the same "Communication" type, we considered them one coding unit. In cases where a single message conveyed more than one "Communication" type, coders were instructed to apply the label that best captured the message's dominant intent.

4.2.4 Inter-Rater-Reliability (IRR). Three annotators (an author, a professional software engineer, a computer science Masters' student) applied the codebook to all 108 chats, with at least two annotators in each of 99/108 = 91.7% of chats. We calculated *inter-rater reliability* (IRR) using the *balanced F-measure* [43] with *relaxed-text-spans* [80], both established methods suitable for our data's lack of defined negative cases [43], variable numbers of annotators [43], and linguistic filler [80]. We calculated IRR for each top-level code family and the 46 lower-level codes, averaging over chats and annotator pairs [43]. All five high-level codes have IRR above 80% (with some up to 95%). Among lower-level codes, 70% (32/46) exceed 50% IRR. Such values are common for fine-grained distinctions (e.g., "suggest" vs. "teach + extensions") and are acceptable when carefully interpreted [57].

4.3 Quantitative Analysis Methodology

We conduct a supervised quantitative analysis of the annotated chats using *generalized linear multi-level regression* (GLMER) modeling with log links. This approach allows us to analyze the occurrence rates of our qualitative codes and relate them to issue-level variables ("Issue Request" and "Issue Outcome"). While we model different features to address each research question, our general methodology is consistent across all analyses.

4.3.1 Suitability of GLMER analysis. Our data may exhibit non-independence between observations (annotated coding units) of the same annotator, chat, speaker type (Learner/Helper), or request type, and thus do not satisfy the assumptions of a simple regression model. GLMER analysis is well-suited for such heterogeneity [9, 27, 33], allowing us to control for the effects of nuisance variables by leveraging random effects to specify potential grouping structures in our model. In addition, GLMER analysis accommodates unbalanced group sizes, which is useful given that not all annotators viewed every chat.

4.3.2 Model structure and inference. We investigate the per-speaker, per-issue frequencies of each code, as well as the frequencies of issue-level attributes ("Issue Request" and "Issue Outcome"). To do so, we construct a dataset where each observation is defined by multiple mathematical model variables: the `code`, the `speaker` (Learner/Helper), the unique identifiers of the surrounding `chat` and `issue`, and the `annotator`'s unique identifier. We model `counts` (the number of times one annotator assigned one code to messages sent by one speaker in one issue) as our response variable. Other explanatory variables include the issue's `outcome` (Boolean, Section

"Communication" Axis Labels	Definition	Individual Codes (separated by ●)
Question Goal ?	Describes the intent of the question.	info gathering ● check if correct ● guiding ● check if following ● personal
Specificity ? ✓	Describes degrees of message specifics.	vague ● detailed
Content Domain ? ✓	Describes the topic of conversation in the message. Topics focus on programming and range from source code to defects to rapport.	proposed new code ● specifications ● original code ● bug ● coding concept ● platform related ● personal info ● test cases ● coding experience ● development strategy ● code opinion ● learning resources ● error location ● error message
Confidence ✓	Describes degrees of conveyed certainty.	certain ● uncertain
Mechanism ✓	Describes the manner information is given.	state/observe ● positive confirmation ● explain ● implement ● suggest ● interactive steps ● negative confirmation ● teach + extensions
Expression ‿	Describes the emotional intent of the message. We identify two categories: positive or negative.	greeting ● gratitude ● supporting words ● being lost ● frustration ● apology ● negative self-judgment ● being incorrect ● teaching philosophy

Axis Label Applies To: ?= Questioning, ✓= Answering, ‿= Attitude

Table 1: Six axes of "Communication" codes, collectively comprising 40 individual sub-codes. Each dimension is labeled with a subset of message types ("Questioning"/"Answering"/"Attitude"). Structural/segmentation message types ("Issue Request" and "Issue Outcome") are elided. In each row, individual codes are sorted by decreasing frequency (median across annotators).

4.2.2) and `Issue type` (Section 4.2.2). In all models, we control for issue length (# of annotations, used in log scale as an offset).

We fit five models, structured to analyze Learners (RQ1) questions; question-asking and answer-providing (RQ3); and issue outcomes (RQ5). RQ2 and RQ4 are descriptive and not suited to regression analysis. To define each model, we apply dataset filtering and pooling and design a corresponding model specification formula. We consider relevant observations, variables, and interactions between variables. We recognize that our Boolean `outcome` is one of many possible success metrics; thus, we only include it in the model explicitly considering issue outcomes (RQ5). For each model specification, we fit Poisson [36, Ch. 5.4.3] [82] and negative binomial [31] models (when tractable). We select final models using the Akaike Information Criterion (AIC) measure of model fit [2, 17].

Explanatory variables can be interpreted as modifying the proportion of annotations in an issue which correspond to a ⟨code, speaker⟩ pair. For variables modeled by random intercepts and slopes [8], these are represented by a Best Linear Unbiased Predictor (BLUP), given as a ***predicted value*** along with a 95% coverage ***prediction interval (PI)***. The BLUP represents the additive change to the log rate of occurrence [65]. Thus, the effect for one level of a qualitative variable, with predicted value β and PI = (a, b), means that the ⟨code, speaker⟩ pair is predicted to occur at e^{β} times its marginal expected frequency when that factor level applies, with 95% confidence that this multiplier (the ***rate ratio***) falls between e^{a} and e^{b}. For clarity, we only present statistically significant effects (i.e., PI excludes zero). The replication package has more detail.

5 Results

We present findings from a mixed analysis study of 108 anonymous peer tutoring chats totaling over 40,000 words. We use qualitative coding (Section 4.2) to identify key conversational behaviors and quantitative modeling (Section 4.3) to examine their relationships and predictive patterns. We source quotes by chat number (C#).

"Issue Request" Type	# Issues	# Successful Issues
code writing	57	33 (57.9%)
bug fixing	55	32 (58.2%)
code comprehension	15	10 (66.7%)
code improvement	6	3 (50.0%)
Total	**133**	**78 (58.6%)**

Table 2: Proportion of successful issues per type of request.

5.1 RQ1 — Why do Learners use Python Tutor?

We show that Learners are primarily CS1-level novices (91%) seeking assistance with writing code (43%) or fixing bugs (41%). Learners who are traditional students often use Python Tutor to supplement other resources. In total, our dataset includes 108 unique Learners, each represented by a single chat session with 1–3 issues.

5.1.1 Most Learners are CS1-Level Novices. **The majority of Learners (95/108) are novices, new to computer science or Python.** We evaluate this by manually examining messages tagged with "coding experience" or "personal info" (see Table 1). Novice Learners do at least one of the following: (1) explicitly identify as a novice (n = 15, e.g., *"I am a newbie…"* (C14)), (2) mention lacking knowledge in topics commonly covered in introductory CS courses (n = 10, e.g., *"...I'm struggling with lists and how to use them obviously"* (C83)), or (3) present problems that focus on a common CS1 course topics, per the CS introductory topics identified by Becker and Fitzpatrick [11] (n = 71, e.g., *"[Helper:] what are you trying to do? [Learner:] append data2 dict to data1"* (C54)).

5.1.2 Learners Want Help Writing Code and Debugging. We find that Learners seek help on four problem types (Table 2): "code writing" (42.9% of all requests), "bug fixing" (41.4%), "code comprehension" (11.3%), and "code improvement" (4.5%). These categories reflect core activities in software development and capture the range of challenges that Learners bring to P2P sessions.

A "code writing" request asks for help writing programs or functions (e.g., *"hey i need help defining a function that prints out "Hermione is a wizard in Mathematics""* (C18)). A "bug fixing" request asks for help fixing bugs (e.g., *"I am trying to find out where I went wrong with my code"* (C8) or *"Need help debugging"* (C7)). A

"code comprehension" request asks for help understanding a program's behavior or clarifying concepts (e.g., *"Why does commenting out* `nonlocal s` *cause an error?"* (C3), or *"can you explain me the 27th line"* (C77)). Finally, a "code improvement" request asks about enhancing the efficiency, or quality of functioning code. This includes optimizing big-O runtime or improving variable names (e.g., *"Could you help me make this code more readable?"* (C11)).

The majority (84.2%) of requests are for writing code and debugging. This reflects that Learners are focused primarily on concrete, low-level, implementation-heavy tasks rather than higher-level, more analytical tasks while on Python Tutor. Interestingly, a futher evaluation of the topic of Learners' questions (i.e., "Questioning"-coded messages, see Section 4.2.2), reveals that the topic of "proposed new code" is significantly more frequent than other topics (i.e., "Content Domains"), regardless of the precipitating "Issue Request" type ($\beta = 1.79$, $RR = 6.02$: defined in Section 4.3.2). Together, these results imply that **Learners desire help with writing new code disproportionately compared to other topics.** Writing new code seems to be key motivator for using Python Tutor's P2P chat. This is important because it highlights a key difference between what Learners seek in P2P tutoring—help with new code—and what is often sought in traditional classroom settings (e.g., help with bugs in existing code [1]).

5.1.3 Learners Use Python Tutor as a Secondary Resource. Beyond just being novices, we observe that some Learners are currently enrolled in a structured programming course: 28 Learners explicitly indicate that they are students by referring to their coursework or academic year (e.g., *"I am just taking CPE 101"*). We investigate why Learners choose to augment course instruction with Python Tutor's unmoderated chat by manually examining messages tagged "personal info". Understanding this choice can provide insight into the specific needs of students seeking help outside the classroom. **We find that Learners describe challenges with the structured instruction available through their traditional courses**: *"Yeah I'm struggling a lot with it, I went to office hours today and they somehow only confused me more"* (C78). Some Learners cite looming deadlines. For example, *"it's due at my school today but they never taught us how to make an input into a list"* (C34). These patterns suggest that some students turn to peer platforms like Python Tutor when formal support falls short. We saw no unsolicited praise of Python Tutor, only critiques of classroom instruction, highlighting both gaps in CS1 support (e.g., office hours not helping) and opportunities for better P2P platform design.

> ***RQ1 — Why do Learners use PythonTutor?***
>
> Almost all Learners are beginners at the CS1 level (95/108) and some identified as traditional students (28/108) with these students more frequently using Python Tutor as a secondary learning resource. Overall, Learners primarily desire help with code writing (43%) and bug fixing (41%).

5.2 RQ2 — Characterizing Helpers

We find that while some Helpers ($n = 21/266$) explicitly or implicitly describe their programming skills (ranging from novice to experienced), most do not state their background directly. Nonetheless, a subset of Helpers ($n = 25$) articulate clear teaching philosophies about how they want to support Learners. Further, Helpers use three communication strategies more frequently than learners, to help achieve these goals. Our findings highlight that informal, volunteer-driven peer tutoring can foster pedagogical behaviors, even without formal training.

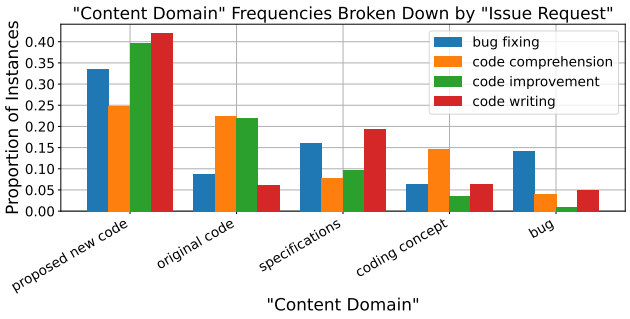

Figure 4: Distribution of the top 5 "Content Domains" (e.g., "bug") as a function of "Issue Request" (e.g., "bug fixing").

5.2.1 Helper Programming Experience Is Diverse but Rarely Specified. Some Helpers reflect a broader range of programming experience than Learners, based on manual inspection of messages coded as "coding experience" or "personal info". **We observed three categories of self-described experience: novice, experienced, and implicit experience.** Ten Helpers identified as beginners (e.g., *"…I am new to python"* [C97, C76]), and one was a high school student. Others expressed more confidence, including one *"python engineer"* (C80), someone who had *"been doing all this from quite sometime now"* (C11), and two who were self-taught (e.g., *"I learned python by myself"* (C16)). Five Helpers revealed their experience level indirectly, often by noting unfamiliarity or uncertainty (e.g., *"never seen except ValueError"* (C8); *"I'm not sure if I can fix it"* (C2)). Overall, Python Tutor draws Helpers from a wide range of backgrounds, some of whom are still learning, themselves.

5.2.2 Some Helpers Express Teaching Philosophies. While most teaching beliefs are expressed implicitly (e.g., by supplying "interactive steps" for problem-solving), surprisingly, **some Helpers explicitly describe their teaching philosophies during interactions with Learners or other Helpers** ($n = 25$). We defined "teaching philosophy" (a sub-code of "Content Domain") as a communication about what the Helper believed the Learner should take away from the conversation. These fell into three themes: (1) encouraging learning alongside correctness (e.g., *"hopefully the solution makes snese to you and is understandable and readable"* (C4)); (2) promoting Learner independence (e.g., *"I'm not here to do your homework for you. That is for you to do"* (C12); *"i suggest you rewrite your question and tell us what you are trying to do instead of trying to fix a subproblem"* (C44]); and (3) emphasizing the importance of practice (e.g., *"But if you practice enough you should be fine!"* (C91)). These moments of explicit pedagogy suggest that some Helpers see themselves not just as problem-solvers, but as informal educators.

5.2.3 Three Helper-Specific Strategies and Varied Participation. Instructional strategies for professors and TAs are well studied, but

peer learning (especially in informal online settings) introduces unique dynamics that are less understood [7, 18, 28, 40]. **In our qualitative analysis of Helpers' responses, beyond providing code implementation help, we observed three relevant strategies:** "guiding" questions (e.g., *"Look at the outputs of the examples on lines 7 and 8 :) What data type is the output?"* (C53), "interactive steps" (e.g., Helpers guide the Learners code writing: *"so step one should be getting a list of those numbers"* (C12), *"try to use paper and pencil to map it out first"* (C48)), and "teach + extensions" (i.e., referencing detail beyond the scope of the current help request). For example, *"[Helper]: Python is actually awesome and has this great built-in string operator "*" that lets you duplicate a string x times"* (C12), where the Helper describes additional coding concepts associated with Python. Across all annotations, we predominantly observed these codes from Helpers, not from Learners. The examples above illustrate how some Helpers adopt pedagogical approaches when assisting peers (see Section 5.2).

Although some helpers engaged deeply, many contributed briefly: of 266 Helpers, 247 sent more than just a greeting, but only 91 sent ten messages or more (average across annotators). This drop-off suggests that many helpers offered transient support, while a subset engaged more consistently. Despite this variance, 58.6% of interactions (Table 2) were rated successful by Learners. In some cases, success involved a core Helper taking a leading role. For example, one Helper might guide the explanation while others stepped back: *"ok I leave you, you are the captain here [Helper], [Learner] listen to him/her ok?"* (C53).

> ***RQ2 — How are Helpers characterized on Python Tutor?***
>
> Unlike Learners, who are mostly novices, some Helpers indicated more diverse experience levels. A subset (n=25) expressed teaching philosophies emphasizing deeper learning, independence, and practice. Lastly, Helpers often used strategies like guiding questions and incremental steps. However, many Helpers contributed only briefly.

5.3 RQ3 — Breaking Down Interactions

Having analyzed Learners (Section 5.1) and Helpers (Section 5.2) separately, we now consider their interactions: the content of chats. Specifically, we find that they overwhelmingly focus on writing code, regardless of the Learner's request. Excitingly, we find a pattern of common software development processes that emerge organically in conversation. All analyses focus on "Questioning" and "Answering" messages.

5.3.1 Emphasis on New Code in All Chats and Specs-Focused Bug Discussions. Out of 14 distinct "Content Domains" identified (see Table 1), **"proposed new code" overwhelmingly dominates conversations across all types of issues** ($\beta = 2.57$, $RR = 13.02$). We obtain this result by examining the distribution of "Content Domains" (see Table 1) via GLMER modeling (see Section 4.3). This is a notable contrast to real-world software engineering, where reading and understanding existing code are significantly more common than writing new code [12, 58, 62, 71]. **This pattern may indicate a disconnect between the Learner's initial question and the Helper's emphasis on code synthesis as a help strategy.** For example, in the excerpt below, a Learner asks a question about Python scoping semantics, but the Helper quickly redirects the conversation to a suggestion to use a data structure: *"[Learner]: Why does commenting out nonlocal s cause an error? ... [Helper]: instead using list i recommend using dict [Learner]: this is a question from an exam. im having trouble understanding why s cannot be called from parent f1"* (C3). The Helper misinterprets the conceptual question as a prompt for implementation advice.

Content Domain	Mechanism	Pred. Val, β	2.5% P.I	97.5% P.I.	Rate Ratio
proposed new code	implement	2.04	0.89	3.18	7.66
coding concept	teach + extensions	1.66	0.44	2.88	5.26
learning resources	teach + extensions	1.41	0.08	2.73	4.08
development strategy	suggest	1.40	0.28	2.53	4.07
bug	explain	1.27	0.16	2.38	3.56
learning resources	suggest	1.23	0.05	2.41	3.42

Table 3: "Content Domains" and "Communication Mechanisms" likely to occur together: the pair's frequency is significantly different from zero, even after controlling for the two marginal frequencies. Statistics defined in Section 4.3.2.

Contrary to this common Helper assumption, we find that P2P interactions are not one-size-fits-all: with statistical significance, the strategies that participants use to communicate with one another are related to the subjects they wish to communicate (see Table 3, showing analysis of "Questioning" and "Answering" messages). In particular, messages discuss "proposed new code" via direct "implementing" at 7.66× the rate that would be expected if the "Content Domain" and "Communication Mechanism" axes (Figure 3) were independent. Meanwhile, "coding concepts" and "learning resources" are disproportionately communicated via "teach + extensions". Participants may naturally find particular communication strategies better for some topics, motivating investigation into whether these pairings encourage successful P2P tutoring.

We also found significantly lower frequencies of "error location" and "error message" ($\beta = -1.58$, $RR = 0.21$), across all request types. This is important since "bug fixing" is the second most common request. For instance, even when Learners explicitly asked for help debugging, conversations often shifted away from talking about defects: *"[Learner]: i'm supposed to change the code so it prints out those results, but without adding any new lines. [Helper]: yes but don't you want match to count not always equal to one but count how many [Learner]: so I should do match = attempt? [Helper]: no it's shoud count like count = count +1"* (C7).

5.3.2 Software Development Patterns Are Common in Chats. Building on patterns in our codebook (Section 4.2.2), we identify four software development stages adapted from the Waterfall model [66]: **Requirements** (e.g., *"[Learner]: I'm not suppose to print anything at the final"* (C7)), **Design**, **Implementation**, and **Verification** (e.g., *"[Helper]: do you have test cases? [Learner]: attempt is [28, 49]..."* (C7)). Though inspired by the Waterfall model, these stages also reflect broader software practices [3, 23]. Our goal was to surface educationally-relevant stages, not enforce a specific methodology. To verify these inferences, one researcher manually reviewed all "Questioning" and "Answering" messages.

Excitingly, 83% of chats (90/108**) included at least one of these four stages, and, very surprisingly,** 30 **chats included and worked through all four stages.** The most frequent stage

was *Requirements*, appearing in 88 chats, which aligns with some software engineering conventional wisdom about where the most costly mistakes are made and how defects are actually defined (cf. [81]). Perhaps surprisingly, the least common stage was *Verification*, found in only 51 chats, which aligns with our observation that corresponding "Content Domains" (e.g., "test cases") were not a focus of P2P tutoring chats (see Figure 4). This analysis suggests that while novice programmers frequently engage in discussions related to code specifications, they do not prioritize testing and verification in these chats. Given the importance of testing in software development, this gap highlights an area where novice programmers may benefit from further instruction and practice.

RQ3 — What is the content of Python Tutor interactions?

First, new code writing was the most common topic, even for conceptual requests. Second, "teach + extensions" (i.e., going above and beyond) was a popular communication strategy for "coding concepts" and "learning resources", even though P2P tutors are untrained and uncompensated. Third, 83% of chats included at least one classic SE development stage (requirements, design, implementation, or verification) and 28% had all four.

5.4 RQ4 — Assessing Emotional Tone

We analyzed emotional attitudes and personal connections, which have been identified as important to outcomes and retention [6, 7, 35]. We find that our unmoderated chats are largely positive.

5.4.1 Users Largely Express Positive Attitudes. In our data, "Expressions" fell into two categories: positive (i.e., fostering a sense of community, optimism, or welcome) and negative (i.e., creating tension or conveying frustration), see Table 1.

We identified five positive expression patterns: "apologize" (i.e., apologizing for a perceived mistake: *"im sorry i understand now"* (C12)), "express being wrong" (i.e., admitting what they said prior was incorrect: *"oh sec my bad"* (C34)), "supporting words" (e.g., *"d('.')d)"* (C12), symbolizing two thumbs up, *"you are very good"* (C9)), "gratitude" (e.g., *"Thank you so much for helping"* (C16)), and "greeting" (e.g., *"Hello! :)"* (C16)). **We find three patterns of negative expressions:** "being lost" (e.g., *"huh?"* (C10, C24) or *"im so confused"* (C78)), "frustration" (e.g., *"Mannn this shit is frustrating"* (C7) or *"IVE BEEN WORKING ON IT FOR AN HOUR STRAUGHT"* (C27)), or "negative self-judgment" (e.g., *"ive been sitting like an idiot for 13 min"* (C59) or *"im not capable of this"* (C82))

Interestingly, **prevalent attitude expressions vary by request type** (full data in replication package). For example, "code comprehension" requests trigger a particularly high rate of "supporting words" (34.9%, e.g., *"[Learner]: . . . thought that my method work too. but too naive xD, [Helper]: ahahha yeah thats alright"* (C92)). In general, however, we find that "greetings" are the most common form of "Expression" (777 annotated messages out of 2, 197). The second and third most common are "gratitude" ($n = 389$ annotations) and "supporting words" ($n = 367$ annotations). Despite the anonymity and lack of moderation in Python Tutor chats, **the overall tone of peer-to-peer interactions leans positively.** This agrees with traditional classrooms where teachers and tutors often aim for positivity [51, 61, 77]. By contrast, other P2P communities like Stack Overflow can show significantly more negativity [29, 72].

5.4.2 Users Make Personal Connections. We explored whether users formed personal connections during P2P tutoring sessions, using the exchange of personal information (e.g., age, location, or social media handles) via manual inspection of messages labeled as "personal info" as a proxy. While uncommon, such disclosures suggest a degree of trust, familiarity, or social bonding.

We found 18 chats with personal information exchanges. These occurred in three main ways: (1) Learners offered contact information to Helpers after successful interactions, often to request ongoing support outside of Python Tutor ($n = 4$; e.g., *"re you on some other platform, i wanna learn a lot from you"* (C20)); (2) Helpers initiated an exchange of contact info to share additional materials ($n = 2$; e.g., *"I'll give you my Skype, so you can drop me the assignment"* (C32)); (3) either party voluntarily shared or asked for personal details unrelated to the task ($n = 12$; e.g., *"can i ask you, where are you from?"* (C66), *"university of antwerp"* (C32)).

Although uncommon, P2P does foster personal connections but they are limited and further research is needed to explore their influence on users and online tutoring. We consider this noteworthy because personal connections can enhance a sense of belonging and community (cf. [72]), which, in turn, may promote inclusivity—particularly among novice or less-represented [29] Learners, who constitute the majority of users on Python Tutor.

RQ4 — How do people feel during Python Tutor chats?

Most annotated messages are positive in tone (1714/2197) with common expressions like "greetings", "gratitude", and "supporting words". "code comprehension" requests often trigger "negative self-judgment" from Learners and encouragement from Helpers. Personal connections (e.g., sharing contact info) were rare (18/108) but occurred in both directions.

5.5 RQ5 — Modeling Successful Interactions

We examine which conversational features predict successful outcomes, defined as positive Learner feedback (Section 3), using three lenses: (1) request type, (2) individual conversation features, and (3) speaker role (Learner vs. Helper). For (1), contrary to expectations, programming-heavy requests (e.g., "bug fixing") were no more successful than conceptual ones (e.g., "code comprehension"); Fisher's exact test found no significant difference in success rates ($p = 0.92$; Table 2). For (2) and (3), we fit a GLMER model using code frequencies as predictors of success. To reduce sparsity, codes are modeled independently. We report only statistically significant or strongly trending effects (full results in replication package).

5.5.1 Learner Satisfied Outcomes. We identify conversational behaviors associated with issue success (defined in Section 3). Because these behaviors occur prior to outcome resolution, they may reflect mechanisms contributing to success or failure.

The sub-code "gratitude" (defined in Section 5.4.1) is positively associated with success ($\beta = 0.34$, $RR = 1.40$). While this aligns with our outcome definition (Learner-reported satisfaction), it also serves as a check on internal validity: interactions marked by mid-conversation positivity are more likely to end well. Similarly, "implementing" (e.g., a strategy where the user writes or modifies code) is also positively associated with success ($\beta = 0.18$, $RR = 1.20$). This suggests that hands-on support (vs. indirect guidance) may be

Codes *Communication Axis → Sub-Code*	Pred. Val, β	Success When:
Question Goal → personal	0.19	$L \gg H$
Content Domain → coding experience	0.16	
Content Domain → test cases	0.13	
Mechanism → positive confirmation	0.07	(no corr.)
Confidence → certain	-0.05	
Content Domain → proposed new code	-0.06	
Specificity → vague	-0.07	
Expression → greeting	-0.08	
Mechanism → explain	-0.13	
Mechanism → implement	-0.21	
Question Goal → check for following	-0.36	$L \ll H$

Table 4: "Communication" codes where the difference in usage between Learners and Helpers (L − H) is significantly predictive of successful issues. Rows sorted by effect size.

especially effective in P2P environments. In contrast, "being lost" ($\beta = -0.34$, $RR = 0.72$) and "frustration" ($\beta = -0.35$, $RR = 0.71$) are both negatively associated with success. These emotions (defined in Section 5.4.1) may signal trouble early, suggesting a role for real-time monitoring and intervention.

5.5.2 Speaker-stratified effects. Beyond overall code frequencies, we examine whether the balance of behaviors between speakers (Learners and Helpers) predicts "Issue Outcome". Specifically, we compute the difference in usage per code (Learner frequency − Helper frequency) and investigate how this difference predicts outcomes. This speaker-stratified analysis identifies 12 codes with statistically significant associations (Table 4), revealing that *who* expresses a behavior can matter more than whether it occurs at all. We control for the marginal effects in Section 5.5.1, e.g., that "implementing" (regardless of the speaker) is predictive of outcome.

"Content Domain": Learners feel more positively about issues where they contribute more discussion of "coding experience" and "test cases", while Helpers take a larger share of the discussion around "proposed new code".

"Mechanism": Successful interactions are marked by more "positive confirmation" from Learners and more "implementing" and "explaining" from Helpers.

"Questioning": We find that successful interactions tend to include more "personal" questions from Learners (e.g., inquiring about each other's experiences) and more "checking for following" questions from Helpers. Successful interactions tend to include Helpers whose messages are more often "vague" (i.e., higher-level or open-ended) and phrased with greater "certainty".

Most notably, three codes: "personal questions", "coding experience", and "test cases", exhibit a striking difference-in-differences pattern. In successful issues, Learners use these codes more than Helpers. In unsuccessful issues, the opposite is true. **This reversal suggests that not just the presence of these behaviors, but the speaker producing them, is tightly tied to success.** These findings offer a clear signal for future design interventions and validate the importance of considering speaker role.

Finally, some significant codes may reflect structural patterns in the platform rather than conversational strategy. For instance, Learners may issue more "greetings" in failed interactions as potential Helpers join and leave without assisting. While still informative, these cases may reflect external dynamics rather than speaker-driven effects.

RQ5 — What leads to successful Python Tutor interactions?

Success is not significantly tied to issue request type. Instead, we find that "implementing" is positively associated with success, while "being lost" and "frustration" are negative indicators. Outcomes also depend on *who* enacts certain behaviors: in successful issues, Learners more often ask personal questions, discuss coding experience, and mention test cases, whereas in failed issues, Helpers do so more (statistically significantly).

6 Discussion

Our findings show that anonymous P2P tutoring can offer real help with real code. We reflect on its value in this generative AI era and outline design opportunities for future systems.

Implications for P2P Tutoring. Our results indicate that P2P tutoring could serve as a scalable supplement to traditional instructional support, especially in large CS1 courses where resources may be stretched thin [70]. P2P systems can help Learner demand with minimal institutional overhead, while supporting a positive tone and maintaining engagement with formal software engineering processes. We find that Helpers encourage active problem-solving skills through varied communication strategies. This may help build critical thinking skills that risk being lost when students over rely on generative AI tools [83]. The diverse experiences, goodwill, and a positive learning atmosphere suggests P2P tutoring is not only viable but valuable, even in the generative AI age.

Design Directions for Future P2P Platforms. First, successful conversations were often marked by role-specific behaviors: Learners asking personal questions or discussing test cases, Helpers confirming that Learners are following, etc. (Section 5.5.1). P2P systems could scaffold users do so more frequently. One promising avenue is an *AI-powered conversational scaffolding agent* that suggests role-specific prompts mid-chat, such as "Did that make sense?" or "Could we walk through a test together?" Second, while toxicity was rare in our dataset, anonymous platforms are vulnerable to abuse. Human moderation is likely too costly; instead, AI moderation agents could flag problematic behavior and intervene early. Third, matching between Learners and Helpers could be improved. Intelligent matching based on programming experience, shared code, communication style, or prior collaboration could lead to more productive sessions and better rapport.

7 Related Work

We situate our work within prior literature on P2P tutoring.

Peer-to-Peer Tutoring. P2P tutoring can effectively help students learn (see Silva *et al.* for a review [74]). Traditionally, P2P tutoring involves students working together in a formal educational setting (e.g., students reviewing each other's work, etc.). Interactions in these setting are frequently studied [53, 56]. P2P participants have typically interacted before the session (e.g., classmates), and

tutors often have incoming training or scaffolding (e.g., TA training [67], etc.). P2P tutoring can improve attitudes and performance in traditional introductory programming classrooms [34].

Real-time collaboration also appears in pair programming, where one student writes code and another reviews and guides it. Though not P2P tutoring per se, pair programming has been studied in educational contexts [76, 78]. For example, Shi *et al.* [73] found that game-based pair programming promoted planning behaviors. However, unlike our work, these studies did not examine anonymous interactions or communication structure.

P2P tutoring has also been effectively adapted to *online* contexts [18, 21, 25, 39, 87]. Closest to our work, Guo *et al.* analyzed a set of *private* P2P chats on Python Tutor where students knew each other before the session [39]. They found that 60% of sessions were collaborative learning activities without a tutoring goal, along with an emphasis on remembering, applying and understanding, but not analyzing, evaluating or creating. By contrast, we observed a significant focus on a tutoring goal, communication strategies, and writing new code. Within the same platform, this difference can be attributed to our focus on *public* chats, where the users are anonymous. We argue that the public aspect of our dataset is essential for deploying P2P tutoring for non-traditional novices at scale. It is crucial to understand the difference in these contexts.

Asynchronous Peer Interactions. To the best of our knowledge, we are the first to explore the dynamics in synchronous, anonymous, unmoderated P2P tutoring. By contrast, both wiki editing [54] and forum posting [86] have been evaluated as learning activities in an introductory computer science course, and found to promote engagement and learning under structured conditions. Studies have also examined Stack Overflow as a learning tool [14, 24]. Swillus *et al.* used mixed-methods to analyze sentiment (similar to "Attitude" in our codebook) in Stack Overflow comments [75], finding that negative sentiments (e.g., insecurity) interact with project complexity and the experiences of engineers. However, they did not analyze conversation topics or how peer interactions relate to an outcome.

AI for Tutoring. An alternative to P2P tutoring leverages AI models like ChatGPT (see meta analysis by Francisco *et al.* [30]). Whether these methods help students is an ongoing research topic. One qualitative study of responses to programming-related questions from Stack Overflow and from ChatGPT and found that programmers of varied experiences preferred the Stack Overflow responses [46]. Regardless, there is a rise of students using AI models to help them learn and understand code [20, 32, 64]. One promising angle for AI in our context is our finding that certain behaviors are more correlated with success when carried out by Learners. Since current AI interfaces (e.g., ChatGPT) are driven by questions and prompts from Learners, AI tools may avoid the "mistake" of asking personal questions or probing for test cases too frequently.

8 Limitations and Threats to Validity

Generalizability. Our data comes from a single platform, which limits external validity. However, Python Tutor is globally accessible and has real-time features that make it a strong case for scalable peer tutoring. Although we mitigate this threat by randomly sampling chats, data were collected prior to the widespread use of LLMs like ChatGPT (cf., 2019), and user norms may have since changed.

Satisfaction as Measure of Outcome. Rather than evaluating direct learning gains, we focus on satisfaction as a proxy for session success: critical factors in persistence and early CS confidence. This captures important Learner outcomes like motivation, but we acknowledge it may not fully reflect learning or long-term understanding. However, this emphasis aligns with the goals of scalable tutoring systems that prioritize emotional support and accessibility over correctness alone (see Section 3). To avoid misinterpretation, we only used it when relevant and interpreted it cautiously.

Qualitative Analysis. We constrained annotation to one "Communication" type per message. Although some utterances could contain multiple intents, allowing multiple overlapping codes may have reduced inter-rater agreement. Instead, annotators were instructed to select the most salient communicative purpose and reached consensus through discussion.

Quantitative Modeling. We used regression analysis to examine relationships between codes and outcomes, which necessarily reduces qualitative richness to discrete labels [69]. To mitigate this, we applied best practices for mixed-methods research: strong inter-rater reliability, careful category pooling, and checks on assumptions (e.g., role labeling, outcome coding). Our use of GLMER models also controls for annotator-level variability. Still, some modeling assumptions (e.g., distributional choices, independence of codes) may imperfectly reflect real-world conversations.

9 Conclusion

We present the first study of online, anonymous, unmoderated, and untrained P2P programming tutoring. We analyzed 108 text-chats comprising over 40,000 words to uncover how novice programmers of various backgrounds interact in the context of coding help. We find a clear partition of user roles: Learners, who are mostly CS1-level novices (95/108) seeking support for writing new code, and Helpers, who are volunteers ranging from high school students to professional engineers. Despite the lack of formal training, many Helpers adopt mentor-like strategies and frequently support Learners beyond simple answers. Surprisingly, 83% of chat sessions organically covered software development stages, including testing and verification. Overall, conversations maintained a positive tone with several conversational behaviors significantly correlating with higher Learner satisfaction. For example, implementation-focused support was broadly helpful, while the asking of personal questions, discussions of code experience, and talking about test cases are more effective when initiated by the Learner rather than the Helper. Taken together, these findings highlight how P2P tutoring, even in unmoderated and anonymous settings, can deliver meaningful support for novice programmers. These exciting results suggest avenues for peer tutor training or AI tool intervention, which may become increasingly important as traditional and non-traditional CS students alike turn to P2P tutoring.

Acknowledgments

This work was supported in part by NSF Award #2211749. We thank our annotators, Amaris Sim and Wenxin He, for their time, care, and thoughtful contributions, without which this would not have been possible. We are grateful to Philip Guo for providing the

Python Tutor dataset. Finally, the first author thanks Jack Mucciaccio for his unwavering support throughout this project.

References

[1] Marzieh Ahmadzadeh, Dave Elliman, and Colin Higgins. 2005. An analysis of patterns of debugging among novice computer science students. In *SIGCSE Conference on Innovation and Technology in Computer Science Education*. 84–88.

[2] Hirotugu Akaike. 1973. Information theory and an extension of the maximum likelihood principle. *International Symposium on Information Theory* (1973), 267–281.

[3] Samar Al-Saqqa, Samer Sawalha, and Heba Abdelnabi. 2020. Agile Software Development: Methodologies and Trends. *Int. J. Interact. Mob. Technol.* 14 (2020), 246–270. https://api.semanticscholar.org/CorpusID:225548331

[4] Rayed AlGhamdi. 2025. Bridging learning gaps through Discord: peer-to-peer learning in computer graphics education. *Learning and Teaching in Higher Education: Gulf Perspectives* (2025).

[5] Stephanie Andel, Triparna de Vreede, Paul E. Spector, Balaji Padmanabhan, Vivek K. Singh, and Gert-Jan de Vreede. 2020. Do social features help in video-centric online learning platforms? A social presence perspective. *Comput. Hum. Behav.* 113 (2020), 106505. doi:10.1016/J.CHB.2020.106505

[6] Mehdi Badali, Javad Hatami, Seyyed Kazem Banihashem, Ebrahim Rahimi, Omid Noroozi, and Zahra Eslami. 2022. The role of motivation in MOOCs' retention rates: a systematic literature review. *Res. Pract. Technol. Enhanc. Learn.* 17, 1 (2022), 5. doi:10.1186/S41039-022-00181-3

[7] Jimoh Bakare and Chibueze Tobias Orji. 2019. Effects of reciprocal peer tutoring and direct learning environment on sophomores' academic achievement in electronic and computer fundamentals. *Education and Information Technologies* 24, 2 (2019), 1035–1055.

[8] Dale J Barr, Roger Levy, Christoph Scheepers, and Harry J Tily. 2013. Random effects structure for confirmatory hypothesis testing: Keep it maximal. *Journal of memory and language* 68, 3 (2013), 255–278.

[9] Douglas Bates, Martin Mächler, Ben Bolker, and Steve Walker. 2014. Fitting linear mixed-effects models using lme4. *arXiv preprint arXiv:1406.5823* (2014).

[10] Papia Bawa. 2016. Retention in online courses: Exploring issues and solutions—A literature review. *Sage Open* 6, 1 (2016), 2158244015621777.

[11] Brett A. Becker and Thomas Fitzpatrick. 2019. What Do CS1 Syllabi Reveal About Our Expectations of Introductory Programming Students?. In *Technical Symposium on Computer Science Education*, Elizabeth K. Hawthorne, Manuel A. Pérez-Quiñones, Sarah Heckman, and Jian Zhang (Eds.). ACM, 1011–1017. doi:10.1145/3287324.3287485

[12] Andrew Begel and Beth Simon. 2008. Novice software developers, all over again. In *International Computing Education Research Workshop*, Michael E. Caspersen, Raymond Lister, and Mike Clancy (Eds.). ACM, 3–14. doi:10.1145/1404520.1404522

[13] Bernard Berelson. 2000. Content analysis in communication research. *Media studies: A reader* 200209 (2000).

[14] Trishala Bhasin, Adam Murray, and Margaret-Anne Storey. 2021. Student Experiences with GitHub and Stack Overflow: An Exploratory Study. In *International Workshop on Cooperative and Human Aspects of Software Engineering*. 81–90. doi:10.1109/CHASE52884.2021.00017

[15] Kathryn Bridson, Jeffrey Atkinson, and Scott D Fleming. 2022. Delivering round-the-clock help to software engineering students using discord: An experience report. In *ACM Technical Symposium on Computer Science Education*. 759–765.

[16] John L Campbell, Charles Quincy, Jordan Osserman, and Ove K Pedersen. 2013. Coding in-depth semistructured interviews: Problems of unitization and intercoder reliability and agreement. *Sociological methods & research* 42, 3 (2013), 294–320.

[17] Joseph E Cavanaugh and Andrew A Neath. 2019. The Akaike information criterion: Background, derivation, properties, application, interpretation, and refinements. *Wiley Interdisciplinary Reviews: Computational Statistics* 11, 3 (2019), e1460.

[18] Shekhar Chandra and Shailendra Palvia. 2021. Online education next wave: peer to peer learning. *Journal of Information Technology Case and Application Research* 23, 3 (2021), 157–172. doi:10.1080/15228053.2021.1980848

[19] Preetha Chatterjee, Minji Kong, and Lori Pollock. 2020. Finding help with programming errors: An exploratory study of novice software engineers' focus in stack overflow posts. *Journal of Systems and Software* 159 (2020), 110454.

[20] Eason Chen, Ray Huang, Han-Shin Chen, Yuen-Hsien Tseng, and Liang-Yi Li. 2023. GPTutor: a ChatGPT-powered programming tool for code explanation. In *International Conference on Artificial Intelligence in Education*. Springer, 321–327.

[21] Yan Chen, Walter S Lasecki, and Tao Dong. 2021. Towards supporting programming education at scale via live streaming. *Proceedings of the ACM on Human-Computer Interaction* 4, CSCW3 (2021), 1–19.

[22] James Cook, Krishnaram Kenthapadi, and Nina Mishra. 2013. Group chats on Twitter. In *International World Wide Web Conference*. 225–236. doi:10.1145/2488388.2488409

[23] Mihai Liviu Despa. 2014. Comparative study on software development methodologies. *Database Systems Journal* 5, 3 (2014).

[24] Pierpaolo Dondio and Suha Shaheen. 2020. Is StackOverflow an Effective Complement to Gaining Practical Knowledge Compared to Traditional Computer Science Learning?. In *International Conference on Education Technology and Computers* (Amsterdam, Netherlands). 132–138. doi:10.1145/3369255.3369258

[25] Travis Faas, Lynn Dombrowski, Alyson Young, and Andrew D Miller. 2018. Watch me code: Programming mentorship communities on twitch.tv. *Proceedings of the ACM on Human-Computer Interaction* 2, CSCW (2018), 1–18.

[26] Fabian Fagerholm, Michael Felderer, Davide Fucci, Michael Unterkalmsteiner, Bogdan Marculescu, Markus Martini, Lars Göran Wallgren Tengberg, Robert Feldt, Bettina Lehtelä, Balázs Nagyváradi, et al. 2022. Cognition in software engineering: A taxonomy and survey of a half-century of research. *ACM Computing Surveys (CSUR)* 54, 11s (2022), 1–36.

[27] Julian J Faraway. 2016. *Extending the linear model with R: generalized linear, mixed effects and nonparametric regression models*. CRC press.

[28] Denae Ford, Kristina Lustig, Jeremy Banks, and Chris Parnin. 2018. "We Don't Do That Here": How Collaborative Editing with Mentors Improves Engagement in Social Q&A Communities. In *Conference on Human Factors in Computing Systems*, Regan L. Mandryk, Mark Hancock, Mark Perry, and Anna L. Cox (Eds.). 608. doi:10.1145/3173574.3174182

[29] Denae Ford, Justin Smith, Philip J. Guo, and Chris Parnin. 2016. Paradise unplugged: identifying barriers for female participation on stack overflow. In *Foundations of Software Engineering*, Thomas Zimmermann, Jane Cleland-Huang, and Zhendong Su (Eds.). ACM, 846–857. doi:10.1145/2950290.2950331

[30] Rodrigo Elias Francisco and Flávio de Oliveira Silva. 2022. Intelligent Tutoring System for Computer Science Education and the Use of Artificial Intelligence: A Literature Review. *CSEDU (1)* (2022), 338–345.

[31] William Gardner, Edward P Mulvey, and Esther C Shaw. 1995. Regression analyses of counts and rates: Poisson, overdispersed Poisson, and negative binomial models. *Psychological bulletin* 118, 3 (1995), 392.

[32] Ritesh Ghimire and Asokan Raji. 2024. Use of Artificial Intelligence in Design, Development, Additive Manufacturing, and Certification of Multifunctional Composites for Aircraft, Drones, and Spacecraft. *Applied Sciences* 14, 3 (2024), 1187.

[33] Joshua B Gilbert, James S Kim, and Luke W Miratrix. 2023. Modeling Item-Level Heterogeneous Treatment Effects With the Explanatory Item Response Model: Leveraging Large-Scale Online Assessments to Pinpoint the Impact of Educational Interventions. *J. Educational and Behavioral Statistics* (2023).

[34] Paul Golding, Lisa Facey-Shaw, and Vanesa Tennant. 2006. Effects of peer tutoring, attitude and personality on academic performance of first year introductory programming students. In *Frontiers In Education*. IEEE, 7–12.

[35] Joselyn Goopio and Catherine Cheung. 2020. The MOOC dropout phenomenon and retention strategies. *Journal of Teaching in Travel & Tourism* 21 (09 2020). doi:10.1080/15313220.2020.1809050

[36] Stefan Th Gries. 2013. *Statistics for linguistics with R: A practical introduction*. Walter de Gruyter.

[37] Philip J. Guo. 2013. Online Python tutor: embeddable web-based program visualization for CS education. In *ACM Technical Symposium on Computer Science Education*. 579–584.

[38] Philip J. Guo. 2021. Ten Million Users and Ten Years Later: Python Tutor's Design Guidelines for Building Scalable and Sustainable Research Software in Academia. In *Symposium on User Interface Software and Technology*, Jeffrey Nichols, Ranjitha Kumar, and Michael Nebeling (Eds.). 1235–1251. doi:10.1145/3472749.3474819

[39] Philip J. Guo, Jeffery White, and Renan Zanelatto. 2015. Codechella: Multi-user program visualizations for real-time tutoring and collaborative learning. In *Visual Languages and Human-Centric Computing*. 79–87. doi:10.1109/VLHCC.2015.7357201

[40] Regine Hampel and Ursula Stickler. 2005. New skills for new classrooms: Training tutors to teach languages online. *Computer assisted language learning* 18, 4 (2005), 311–326.

[41] Greg Hart. 2025. *Global Skills Report*. Technical Report. Coursera Enterprise.

[42] Rose Horowitch. 2025. The Computer-Science Bubble Is Bursting. In *The Atlantic*.

[43] George Hripcsak and Adam S Rothschild. 2005. Agreement, the f-measure, and reliability in information retrieval. *Journal of the American medical informatics association* 12, 3 (2005), 296–298.

[44] Katy Jordan. 2014. Initial Trends in Enrolment and Completion of Massive Open Online Courses. *International Review of Research in Open and Distance Learning* 15 (02 2014), 133–160. doi:10.19173/irrodl.v15i1.1651

[45] Katy Jordan. 2015. Massive open online course completion rates revisited: Assessment, length and attrition. *International Review of Research in Open and Distributed Learning* 16, 3 (2015), 341–358.

[46] Samia Kabir, David N Udo-Imeh, Bonan Kou, and Tianyi Zhang. 2024. Is Stack Overflow obsolete? an empirical study of the characteristics of ChatGPT answers to stack overflow questions. In *Conference on Human Factors in Computing Systems*. 1–17.

[47] Sal Khan. 2025. *Free online tutoring, with students like you*. Technical Report. https://schoolhouse.world/

[48] Shahedul Huq Khandkar. 2009. Open coding. *University of Calgary* 23, 2009 (2009), 2009.

[49] Sophia Krause-Levy, William G. Griswold, Leo Porter, and Christine Alvarado. 2021. The Relationship Between Sense of Belonging and Student Outcomes in CS1 and Beyond. In *International Computing Education Research*. 29–41. doi:10.1145/3446871.3469748

[50] Kathleen J. Lehman, Kaitlin N.S. Newhouse, Sarayu Sundar, and Linda J. Sax. 2023. Nevertheless, They Persisted: Factors that Promote Persistence for Women and Racially/Ethnically Minoritized Students in Undergraduate Computing. *Computer Science Education* 33, 2 (2023), 260–285. doi:10.1080/08993408.2022.2086401

[51] Mark R Lepper and Maria Woolverton. 2002. The wisdom of practice: Lessons learned from the study of highly effective tutors. In *Improving academic achievement*. Elsevier, 135–158.

[52] Annie Li, Madeline Endres, and Westley Weimer. 2022. Debugging with stack overflow: Web search behavior in novice and expert programmers. In *International Conference on Software Engineering*. 69–81.

[53] Yinmiao Li, Haoqi Zhang, and Eleanor O'Rourke. 2024. The Undervalued Disciplinary and Emotional Support Provided By Teaching Assistants in Introductory Computer Science Courses. In *International Conference of the Learning Sciences*. International Society of the Learning Sciences.

[54] Yu-Tzu Lin, Cheng-Chih Wu, and Chiung-Fang Chiu. 2018. The use of wiki in teaching programming: Effects upon achievement, attitudes, and collaborative programming behaviors. *International Journal of Distance Education Technologies (IJDET)* 16, 3 (2018), 18–45.

[55] Bo Liu and David J. Malan. 2024. Teaching CS50 with AI. In *Technical Symposium on Computer Science Education*. 1150–1156. doi:10.1145/3626252.3630833

[56] Julia M. Markel and Philip J. Guo. 2021. Inside the Mind of a CS Undergraduate TA: A Firsthand Account of Undergraduate Peer Tutoring in Computer Labs. In *Technical Symposium on Computer Science Education*. 502–508. doi:10.1145/3408877.3432533

[57] Nora McDonald, Sarita Schoenebeck, and Andrea Forte. 2019. Reliability and Inter-rater Reliability in Qualitative Research: Norms and Guidelines for CSCW and HCI Practice. *Proc. ACM Hum.-Comput. Interact.* 3, CSCW, Article 72 (Nov. 2019), 23 pages. doi:10.1145/3359174

[58] Roberto Minelli, Andrea Mocci, and Michele Lanza. 2015. I know what you did last summer — an investigation of how developers spend their time. In *International Conference on Program Comprehension*. 25–35.

[59] Kenrick Mock. 2019. Experiences using Discord as platform for online tutoring and building a CS community. In *Technical Symposium on Computer Science Education*. 1284–1284.

[60] Markus Nivala, Alena Seredko, Tanya Osborne, and Thomas Hillman. 2020. Stack overflow–informal learning and the global expansion of professional development and opportunities in programming?. In *2020 IEEE Global Engineering Education Conference*. 402–408.

[61] Mia O'Brien and Levon Blue. 2018. Towards a positive pedagogy: designing pedagogical practices that facilitate positivity within the classroom. *Educational Action Research* 26, 3 (2018), 365–384.

[62] Thomas M. Pigoski. 1996. *Practical Software Maintenance: Best Practices for Managing Your Software Investment* (1st ed.). Wiley Publishing.

[63] Md Rahman and Yutaka Watanobe. 2023. ChatGPT for Education and Research: Opportunities, Threats, and Strategies. *Applied Sciences* 13 (05 2023), 5783. doi:10.3390/app13095783

[64] Jaakko Rajala, Jenni Hukkanen, Maria Hartikainen, and Pia Niemelä. 2023. Call me Kiran — ChatGPT as a Tutoring Chatbot in a Computer Science Course. In *International Academic Mindtrek Conference*. 83–94.

[65] George K Robinson. 1991. That BLUP is a good thing: the estimation of random effects. *Statistical science* (1991), 15–32.

[66] Winston W Royce. 1987. Managing the development of large software systems: concepts and techniques. In *Proceedings of the 9th international conference on Software Engineering*. 328–338.

[67] Suzanne M Ruder and Courtney Stanford. 2020. Training undergraduate teaching assistants to facilitate and assess process skills in large enrollment courses. *Journal of Chemical Education* 97, 10 (2020), 3521–3529.

[68] Adrian Salguero, William G. Griswold, Christine Alvarado, and Leo Porter. 2021. Understanding Sources of Student Struggle in Early Computer Science Courses. In *International Computing Education Research*. 319–333. doi:10.1145/3446871.3469755

[69] Margarete Sandelowski. 2001. Real qualitative researchers do not count: The use of numbers in qualitative research. *Research in nursing & health* 24, 3 (2001), 230–240.

[70] Linda J. Sax, Kathleen J. Lehman, and Christina Zavala. 2017. Examining the Enrollment Growth: Non-CS Majors in CS1 Courses. In *Technical Symposium on Computer Science Education*. 513–518.

[71] Robert C Seacord, Daniel Plakosh, and Grace A Lewis. 2003. *Modernizing legacy systems: software technologies, engineering processes, and business practices*. Addison-Wesley Professional.

[72] Subhasree Sengupta and Caroline Haythornthwaite. 2020. Learning with Comments: An Analysis of Comments and Community on Stack Overflow. In *Hawaii International Conference on System Sciences*. 1–10. https://hdl.handle.net/10125/64096

[73] Joshua Shi, Armaan Shah, Garrett Hedman, and Eleanor O'Rourke. 2019. Pyrus: Designing a collaborative programming game to promote problem solving behaviors. In *Proceedings of the 2019 CHI conference on human factors in computing systems*. 1–12.

[74] Leonardo Silva, António José Mendes, and Anabela Gomes. 2020. Computer-supported collaborative learning in programming education: A systematic literature review. In *2020 IEEE Global Engineering Education Conference (EDUCON)*. IEEE, 1086–1095.

[75] Mark Swillus and Andy Zaidman. 2023. Sentiment overflow in the testing stack: Analyzing software testing posts on Stack Overflow. *Journal of Systems and Software* 205 (2023), 111804.

[76] Despina Tsompanoudi, Maya Satratzemi, and Stelios Xinogalos. 2015. Evaluating the effects of scripted distributed pair programming on student performance and participation. *IEEE Transactions on education* 59, 1 (2015), 24–31.

[77] Jacqueline Ullman. 2019. Teacher positivity towards gender diversity: Exploring relationships and school outcomes for transgender and gender-diverse students. In *Gender and Sexuality in Education and Health*. Routledge, 42–55.

[78] Karthikeyan Umapathy and Albert D Ritzhaupt. 2017. A meta-analysis of pair-programming in computer programming courses: Implications for educational practice. *ACM Transactions on Computing Education (TOCE)* 17, 4 (2017), 1–13.

[79] Arizona State University. 2024. *ASU announces collaboration with OpenAI to explore use of ChatGPT in teaching and learning*. https://news.asu.edu/20240118-asu-announces-collaboration-openai-explore-use-chatgpt-teaching-and-learning Accessed: 2025-09-28.

[80] Kanix Wang, Robert Stevens, Halima Alachram, Yu Li, Larisa Soldatova, Ross King, Sophia Ananiadou, Annika M Schoene, Maolin Li, Fenia Christopoulou, et al. 2021. NERO: a biomedical named-entity (recognition) ontology with a large, annotated corpus reveals meaningful associations through text embedding. *NPJ systems biology and applications* 7, 1 (2021), 38. Supplemental material.

[81] David Gray Widder and Claire Le Goues. 2024. What Is a 'Bug'? *Commun. ACM* 67, 11 (Oct. 2024), 32–34. doi:10.1145/3662730

[82] Bodo Winter and Paul-Christian Bürkner. 2021. Poisson regression for linguists: A tutorial introduction to modelling count data with brms. *Language and Linguistics Compass* 15, 11 (2021), e12439.

[83] Yuankai Xue, Hanlin Chen, Gina R. Bai, Robert Tairas, and Yu Huang. 2024. Does ChatGPT Help With Introductory Programming? An Experiment of Students Using ChatGPT in CS1. In *International Conference on Software Engineering: Software Engineering Education and Training*. 331–341.

[84] Chunpeng Zhai, Santoso Wibowo, and Lily D Li. 2024. The effects of over-reliance on AI dialogue systems on students' cognitive abilities: a systematic review. *Smart Learning Environments* 11, 1 (2024), 28.

[85] Yan Zhang and Barbara M Wildemuth. 2009. Qualitative analysis of content. *Applications of social research methods to questions in information and library science* 308, 319 (2009), 1–12.

[86] Joyce Zhu, Jeremy Warner, Mitchell Gordon, Jeffery White, Renan Zanelatto, and Philip J Guo. 2015. Toward a domain-specific visual discussion forum for learning computer programming: An empirical study of a popular MOOC forum. In *Symposium on Visual Languages and Human-Centric Computing*. 101–109.

[87] Irina Zinovyeva, Volodymyr Artemchuk, Anna Iatsyshyn, O Popov, Kovach Valeriia, Iatsyshyn Andrii, Y Romanenko, and O Radchenko. 2021. The use of online coding platforms as additional distance tools in programming education. *Journal of Physics: Conference Series* 1840 (03 2021), 012029. doi:10.1088/1742-6596/1840/1/012029