Howzat? Appealing to Expert Judgement for Evaluating Human and AI Next-Step Hints for Novice Programmers*

NEIL C. C. BROWN, King's College London, UK
PIERRE WEILL-TESSIER, King's College London, UK
JUHO LEINONEN, Aalto University, Finland
PAUL DENNY, The University of Auckland, New Zealand
MICHAEL KÖLLING, King's College London, UK

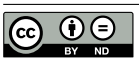
Motivation: Students learning to program often reach states where they are stuck and can make no forward progress – but this may be outside the classroom where no instructor is available to help. In this situation, an automatically generated next-step hint can help them make forward progress and support their learning. It is important to know what makes a good hint or a bad hint, and how to generate good hints automatically in novice programming tools, for example using Large Language Models (LLMs).
Method and participants: We recruited 44 Java educators from around the world to participate in an online study. We used a set of real student code states as hint-generation scenarios. Participants used a technique known as comparative judgement to rank a set of candidate next-step Java hints, which were generated by Large Language Models (LLMs) and by five human experienced educators. Participants ranked the hints without being told how they were generated. The hints were generated with no explicit detail given to the LLMs/humans on what the target task was. Participants then filled in a survey with follow-up questions. The ranks of the hints were analysed against a set of extracted hint characteristics using a random forest approach.
Findings: We found that LLMs had considerable variation in generating high quality next-step hints for programming novices, with GPT-4 outperforming other models tested. When used with a well-designed prompt, GPT-4 outperformed human experts in generating pedagogically valuable hints. A multi-stage prompt was the most effective LLM prompt. According to a fitted random forest model, the two most important factors of a good hint were length (80–160 words being best), and reading level (US grade nine or below being best). Offering alternative approaches to solving the problem was considered bad, and we found no effect of sentiment.
Conclusions: Automatic generation of these hints is immediately viable, given that LLMs outperformed humans – even when the students' task is unknown. Hint length and reading level were more important than several pedagogical features of hints. The fact that it took a group of experts several rounds of experimentation and refinement to design a prompt that achieves this outcome suggests that students on their own are unlikely to be able to produce the same benefit. The prompting task, therefore, should be embedded in an expert-designed tool.

CCS Concepts: • **General and reference** → **Empirical studies**; *Evaluation*; • **Computing methodologies** → *Artificial intelligence*; *Classification and regression trees*; • **Social and professional topics** → **Computer science education**.

Additional Key Words and Phrases: LLMs, AI, Java, Next-step hints, comparative judgement

*Howzat is a contraction of "How's that?", used in cricket to appeal to the umpire [referee] for a decision.

Authors' Contact Information: Neil C. C. Brown, neil.c.c.brown@kcl.ac.uk, King's College London, London, UK; Pierre Weill-Tessier, pierre.weill-tessier@kcl.ac.uk, King's College London, London, UK; Juho Leinonen, juho.2.leinonen@aalto.fi, Aalto University, Espoo, Finland; Paul Denny, paul@cs.auckland.ac.nz, The University of Auckland, Auckland, New Zealand; Michael Kölling, michael.kolling@kcl.ac.uk, King's College London, London, UK.



ACM 1946-6226/2025/1-ART1
https://doi.org/10.1145/3737885

ACM Reference Format:
Neil C. C. Brown, Pierre Weill-Tessier, Juho Leinonen, Paul Denny, and Michael Kölling. 2025. Howzat? Appealing to Expert Judgement for Evaluating Human and AI Next-Step Hints for Novice Programmers. *ACM Trans. Comput. Educ.* 1, 1, Article 1 (January 2025), 43 pages. https://doi.org/10.1145/3737885

1 Introduction

Students who are learning programming often get into a stuck state where they cannot make progress [94]. This may be because they cannot solve a compiler error [78], a run-time error [25, 92], or other more general issues with problem solving [12, 77]. There has been work to try to offer hints to students based on intelligent tutors [15] or crowd-sourced hints [27] or explanations [28], but the new growth of generative Artifical Intelligence (AI) tools offers new possibilities for generating these hints.

Offering hints to students is a subtle art [60, 93]. Just giving the answer offers little or no pedagogical benefit, but being too coy or obscure is not helpful and may frustrate the student further. Choosing the right level of hint is typically more difficult than offering the actual solution. Human teachers have the advantage of often knowing more context about the student and rich knowledge from the student's reaction when attempting an explanation – but teachers are often in large classes and cannot be present at every moment (including when the student works separately outside class) to give hints, so automated approaches to hinting are of interest to provide scale and constant availability.

Generative AI systems such as Large Language Models (LLMs) offer ways to aid students [18], such as generating hints. Students can already directly interact with such LLM systems, but this has two key problems. The first is that students will often ask for the answer, not for a hint, which is less pedagogically beneficial. The second is that students who struggle may be ill equipped to write good prompts to the LLM [17]. Therefore it may be best for a tool, such as an Integrated Development Environment (IDE), to provide the prompt on behalf of the student, in order to generate a hint [75]. It is this approach that we investigate in this paper.

The task we are setting the LLM here is more complex than that of solving a programming problem (which LLMs have been shown to be capable of [44, 58]). The model has to work out what the intended solution is, and it has to do this from limited information: in our context, instead of being given a specification of the programming task, the input consists solely of an erroneous, work-in-progress snapshot of an inexperienced programming novice's source code. The task has to be inferred. When the LLM has deduced the solution, it is expected to *not give it to the student*, but instead devise a pedagogically useful hint that creates a learning experience in which the student makes progress towards a solution.

Generating the hint automatically leads to a set of questions: What makes a good prompt for our purpose? And even with the best prompts and the best LLM: Are the hints generated good enough to be worth showing to novices? Can we consistently generate hints of good enough quality that they help students more than they confuse them? Attempts to investigate these issues reveal two fundamental questions we need to answer as part of this work: What makes a good hint, anyway? And how do we judge whether a hint is "good enough"?

To answer the first of these two questions, we use a method called *comparative judgement* to rank a number of hints (see section 3) according to their quality. This allows us to extract characteristics of good hints in general. To judge whether the hints are of sufficient quality to be shown to students, we use a benchmark: we compare generated hints to those given by expert humans. If the generated hints are judged better than those from humans, they provide an improvement on the status quo and are therefore useful. The details of the methodology are described in section 3.

Our research questions are:

- **RQ1**: Which features of hints (e.g. length, readability) are most important for determining a hint's quality as evaluated by educators?
- **RQ2**: Which generator method (i.e. humans or LLMs) produces the best hints, as determined by educators?
- **RQ3**: Which of a set of candidate prompts produced the best hints, as determined by educators?

This work provides the following major contributions:

(1) We evaluate **which LLM** currently performs best in providing next-step hints for novice programmers.
(2) We determine **the best prompts** for generating hints using the state-of-the-art LLMs at the time of the experiment.
(3) We investigate whether optimal LLM/prompt combinations can **perform as well as** (or better than) **humans** in generating hints.
(4) We describe and demonstrate **a repeatable method** for determining the best prompts and evaluating their performance, which could be re-used when LLM systems update in future.
(5) We investigate **whether comparative judgement is a viable method** for providing rankings in computing education research studies.
(6) We summarise the characteristics of the best hints (as rated by educators) to **determine what makes a good hint**: what length, what kind of language, what pedagogical features.

2 Related work

We divide related work into two main parts: prior work on hint-generation that did not use LLMs, and the use of LLMs in programming education. We also review related work on when hints should be given, as that is a basis for selecting data for our paper. However, we begin with some definitions to help contextualise how the previous work relates to this study.

2.1 Our context: what is a hint?

Terms like feedback and hint are relatively non-specific. In this paper we are interested in the following kind of hint, sometimes referred to as a next-step hint. Imagine that a novice programmer is working alone (for example, at home) on a programming assignment for their study. They are trying to accomplish a goal which is known to them and their teacher, but is not known to the programming system – which might be a general-purpose IDE like VS Code or BlueJ. They get stuck while programming and there is a button to click (or perhaps an automatic popup) that can use AI to give them a **hint**: some text which relates to their current programming attempt (but which must infer the goal), which tells them how to proceed from their stuck state, in a pedagogically useful way (i.e., not just stating what to type into the IDE, but giving instruction) without giving away too much detail such that the student no longer needs to think about the problem. This is the kind of hint we are interested in, although related work may use that term and related terms (such as feedback) using their own definitions. In the KLI framework of Koedinger et al. [47] these kind of hints form both assessment events (is the code right?) and instructional events (what to do next) but we are primarily interested in the latter part, with the assumption that the former is already decided: the code is wrong and the student needs a hint.

We are interested in this specific context as several of the authors are involved in designing tools where these kinds of next-step hints could be built in. Many other designers of these tools will be wondering about adding similar features, and many educators will be keen that such a feature is designed with pedagogy in mind, rather than mere productivity (i.e. just telling the student the answer, or even inserting the code automatically) as would be the case in a professional tool.

2.2 Non-LLM hinting for novice programmers

The idea of giving automated hints to stuck novice programmers has a long history that predates LLMs, and thus there have been multiple reviews on the topic. Crow et al. [15] conducted a review of intelligent programming tutors, from the 1980s to 2018, and found that they were very varied in the features they provide, including whether hinting support is present or not. Keuning et al. [42] performed a systematic literature review of feedback generation in general in programming education. They found that relatively little feedback generation focused on next-step hints. McBroom et al. [63] surveyed hint generation systems from 2014–2018 and introduced a framework that synthesised work on hint generation. Interestingly, it is not clear that LLM-based hinting would fit into McBroom et al.'s framework (which revolves around constructing hints by starting with sets of hint data to narrow down or transform), suggesting that generative AI is quite distinct from existing work on automated hint generation. On a similar note, Marwan and Price [62] describe that data-driven hints (i.e. hints inferred from from previous solutions) typically tell students "what to do, not why", but LLM-generated hints can do both.

In terms of non-AI techniques to generate hints, one way to generate them is to imitate the hints that teachers would give [37, 93]. Another is to use techniques such as program repair to generate fixes [2, 73], or hand-written rules [36, 95]. Existing solutions to a specific programming problem can be used to infer hints for future attempts [70].

Several aspects of automated hinting see no overall agreement in the literature, and evidence is inconclusive or contradictory. While several studies are largely positive about the value of automated hint generation [14, 24, 60, 85], other studies provide some evidence to suggest that hints themselves may not be useful [82], or that they may not help learning or even have a negative effect [62]. One approach to hinting is to show multiple possible hints, at the different appropriate points in the code where each hint could be enacted, although some students found this overwhelming [81] – and similar research on suggested fixes for errors found that students tended not to use the fixes even when they were appropriate [9]. Students report difficulties using hints that are vaguer [81]. Hints which have an explanation are perceived as more helpful and more interpretable but do not necessarily result in better performance or learning [60, 61].

Overall, it appears that hints must be carefully designed in order to be useful. Aleven et al. [3] says "writing good hints is a balancing act," explaining that they need to clear, contain several useful components, yet be short. Marwan et al. [61] found that explanations alongside hints were perceived as more useful and led to better understanding of the hint; Marwan and Price [62] found broadly the same, as did Rivers [85] – who also noted that less experienced students wanted more detail. Price et al. [81] found that withholding the exact edit in a hint led to students being unable to action them, and did not promote reasoning among the students.

2.3 Types of hints

Most of the programming hint literature is concerned with discussing the various ways to automate the activity. However, there is work, both practical and theoretical on what makes a good hint in other domains or more generally. For example, Koedinger et al. [47] provides a framework that can be used as the basis for work in this area, at the intersection of instructional events (which offer explanations) and assessment events (which happen when students receive feedback on their performance).

Aleven et al. [3] suggest that principle-based hints, which "explain – usually over multiple 'levels' of hints – which problem-solving principle applies in the next step of a problem, what the principle says, how it applies, and what concretely to do" are not the best option for most students. Tutored problem solving, which is closer to our hints, may not directly cause learning but may do

so indirectly. Our intention is to avoid what they call "bottom-out" hints which show the solution, in favour of constraining to a single, solutionless hint. We are open to whether hints choose to explain the general principle or focus on the specific solution to the concrete problem at hand.

2.4 Timing of hints

There is a large body of work to identify struggling students across the duration of a whole course [1, 22, 30]. However, this is a distinct problem from trying to identify which students need hints and, crucially, *when* precisely they would benefit from a hint.

A programming environment can provide a hint when students attempt to run the program [73] or when they receive a compiler error [2]. The most common approach is to wait until the student explicitly requests a hint [31, 36, 70, 95]. Alternatively, some tools suggest hints automatically as the user enters code [29, 79].

Shute [91] suggests that feedback should be immediate for activities like programming and mathematics, which supports the idea of having hints inside the programming environment, and also suggests that students will attend more to feedback that occurs when they are certain their answer is incorrect. Conversely, Corbett and Anderson [14] claim that "feedback timing has little consistent impact on learning, although delayed feedback can facilitate retention". Jeuring et al. [37] performed a study asking experts when they would intervene to provide hints, and found "a frequent conflict caused by different pedagogical approaches: (a) an early intervention prevents a student from writing unnecessary code and spending extra time on an assignment, which may lead to student confusion and frustration, versus (b) a delayed intervention gives a student a chance to struggle productively, which may improve student learning." In a follow-up study, Lohr et al. [56] found similarly mixed results over whether educators chose or chose not to provide feedback at specific points: "sometimes one expert uses a reason at a step to explain why they do intervene, and another expert uses the same reason at the step to not intervene." Thus there is no clear recommendation from the literature for how to systematically determine the best time to give a next-step hint during programming activity.

Such a recommendation would have been useful to the present study, to help us choose our snapshots – the points in time where students would most benefit from a hint. In lieu of this recommendation, and in light of the wide range of opinions between experts, we had no choice but to fall back on our own judgement. Note that this is not a critical part of the study: there are no actual students in this study, and the "timing" only affects our choice of the stuck-student snapshots that we picked as scenarios for generating a hint to evaluate.

2.5 Generative AI and Large Language Models in programming education

Since the release of ChatGPT in late 2022 there has been an explosion of interest in LLMs, including in programming education teaching and research. The result has been a dizzying rate of publication; all of the LLM studies cited in this section (of which there are more than thirty) were published in the last two years. In an early 2023 study, educators were found to be split between those who wanted to resist AI tools and those who wanted to embrace them [51]. However, the recent explosion in popularity seems to imply that resistance may well be futile [26, 39], and it may be best to consider adapting our pedagogy [19, 41, 89] and assessment [83] instead. In this section we survey different aspects of LLMs in programming education in turn: students using them directly, students using them via tools, and their use in generating hints and explanations.

2.5.1 Students' direct use of Large Language Models. A natural first step in studying LLMs in programming education was to study what happens when students used LLMs directly, without scaffolding, in a manner of their choosing.

Prather et al. [79] performed a study of how novices worked with Copilot, a generative AI tool that is designed to aid in program construction. They found that novices struggle to understand and use the tool. A later study by Prather et al. [80] suggested that LLMs may widen the divide between students at the top and bottom of the class. Zamfirescu-Pereira et al. [99] observed users without experience in LLM prompts as they designed a chatbot. They found that the users struggled to modify LLM prompts to achieve the desired effect, although they were not using the LLM to directly modify program code. Fiannaca et al. [23] found that users could struggle with what made an effective prompt, and worried about syntax issues within prompts such as line breaks, and the presence or absence of question marks. Denny et al. [17] explored how students write prompts in order to solve programming exercises, when the exercises themselves could not simply be copy-and-pasted into the prompt. They found that "many students, even ones many years into their programming education, do not necessarily understand how to write effective prompts [for LLM systems]." Nguyen et al. [69] similarly found that many novices struggled to write effective prompts when using LLMs for the first time.

Thus the initial research in the area suggests that novices struggle to directly use LLMs in an effective manner. This chimes with other research considering the pedagogical implications: Xue et al. [98] and Kazemitabaar et al. [40] found that direct use of LLMs did not produce any significant effect on learning (although the latter suggest that students with higher prior knowledge may have received greater benefits from using the generator than students with less prior knowledge), while Mailach et al. [59] concluded that "we cannot just give vanilla [LLM] chatbots to students as tools to learn programming, but we additionally need to give proper guidance on how to use them—otherwise, students tend to use it mainly for code generation without further reflection on or evaluation of generated code."

2.5.2 Student use of tools powered by Large Language Models. An alternative mode of use suggested by several researchers [65, 88, 97] is to build tools powered by LLMs. This can avoid issues with students' inability to create prompts, and give more control over the tools' output.

Liffiton et al. [54] created a tool where users could fill in four items: which programming language is being used, the relevant code, the error (if any), and the question they want help with. This is then structured into a single larger prompt to the LLM, and the response is shown in the tool. They concluded that students liked the tool, including the fact that it did not just "give away the answer". In a follow-up study, Sheese et al. [90] found that students tended to ask for help with their immediate problem and would not typically ask more general queries, such as seeking understanding of a wider concept.

Birillo et al. [6] combined LLMs with static analysis in order to create a tool that provides next-step hints. A brief evaluation with students suggested that the tool showed promise. Denny et al. [18] studied students' use of an LLM-powered assistant, and found that students engaged with it extensively, and also found that students preferred receiving scaffolding and guidance rather than simply being told an answer directly. This suggests that hint-generation may be a more useful and more popular tool for students than simply providing correct program code.

2.5.3 Use of Large Language Models for hinting and explanation. Several studies have investigated the use of LLMs for feedback, explanation or hinting – the latter being the precise topic of the current research.

Leinonen et al. [53] used some early LLMs to generate enhanced programming error message explanations. The researchers rated the error message explanations as high quality. More recently, Cucuiat and Waite [16] investigated secondary school teachers' views on LLM-generated programming error message explanations. They used feedback literacy theory to analyse interviews and

found that educators preferred LLM explanations that *guided* and *developed understanding* rather than *tell* (emphasis indicates terms defined by feedback literacy theory [64]).

Hellas et al. [31] investigated the use of LLMs to solve historic student help requests. They used data from a course where students could ask for help from human teachers. They used the code that students asked for help on, combining it with a boilerplate AI prompt, then analysed the responses that came back from the AI, in terms of features such as "identifies at least one actual issue" and "includes code" in order to compare two different AIs (Codex and GPT-3.5), each in English and in Finnish. They found that the AI would frequently provide code despite being instructed not to, and that LLMs could make the same mistakes as students when trying to help them. Roest et al. [86] created a tool to give next-step hints for novice Python programmers, by providing the problem description and current code, and evaluated them with three students and two educators. They similarly found that it was difficult to control the LLM output and that the hints were sometimes misleading. Kiesler et al. [43] also used a similar technique of using historic incorrect student code submissions and analysed the responses using the feedback categorisation of Keuning et al. [42]. They again found LLMs could be misleading but that the quality was generally good.

MacNeil et al. [57] included code explanations generated by LLMs into an online course and found that students rate AI-generated explanations useful for their learning, and that students preferred concise, high-level explanations over line-by-line explanations of the code. Leinonen et al. [52] found that LLMs produced explanations of program code that were rated as more accurate and easier to understand than explanations generated by students on the course. This suggests that AI hints may be very valuable for learners.

Pankiewicz and Baker [72] investigated students' affective states when receiving hints from GPT for solving compiler error messages, and found an increase in focus and decrease in confusion, although more generally they found a mixed pattern as to whether students reported that the results were useful for not, and a mixed result in performance. Xiao et al. [97] investigated students' opinions of using an LLM-powered tool to generate hints. They found that the hints were high quality, but students commented on the lack of flexibility in the interface, and they were confused by some of the higher-level hints which they could find vague. Nguyen and Allan [68] used few-shot learning to train GPT-4 to generate hints and had two instructors evaluate them for accuracy and usefulness, finding that the model performed well and was useful.

Overall, previous studies have suggested that LLMs can produce good quality hints, but there are challenges to tightly control the output (in particular, to avoid giving too much program code or too much of the solution) and avoid misleading hints. As we will detail in the next section, our study differs from previous work in the following ways:

- We focus on stuck students with no information about the problem description that they are working on – a harder but more flexible domain than when the problem is known.
- We use a larger-scale educator evaluation in contrast to the prior evaluations that use 2–3 educators (often the researchers themselves).
- We provide a detailed assessment of hint characteristics in multiple dimensions (length, readability, pedagogy, correctness, sentiment) that combine the disparate subsets into one.
- We ask the educators to use comparative judgement in order to form a ranking of hints which allows us to infer links between these hint characteristics and relative hint quality. This provides useful information about hint attributes that is orthogonal to how they were generated.
- We include human-generated hints to allow us to compare performance of human experts to LLMs.

- We assess multiple prompts with multiple models to see how much effect prompts have on model performance.

3 Method

From our survey of prior work in subsection 2.2, it seems that work on non-AI hints has shown mixed results as to its effectiveness. Many hint approaches rely on knowledge derived from existing solutions to the problem and thus target that specific problem. Generative AI, LLMs in particular, has the potential to be more flexible and powerful when generating a hint for an unseen problem. Studies of LLMs find that students can struggle to formulate prompts, with an alternative approach emerging of using a tool to construct the prompt based on constrained, guided information from the user.

Our approach in this paper is to use source code data from real-world programming sessions from stuck students. We will use the term *Snapshot* for a code sample of a student during a programming session at a point in their work when they were stuck.

We then present each Snapshot to a set of *Generators*. We use the term *Generator* to refer to the producer of the hint: this is either a combination of a specific LLM with a specific prompt, or an individual human.

Next, we show these resulting hints to a set of experienced educators. The educators tell us which hints they consider to be the best. This gives us five results: the best hints, the best-performing LLMs, the prompt templates to generate the best hints, a comparison of human and LLM hints, and the general characteristics of the best hints.

Our method thus has four main parts: the acquisition and selection of the student Snapshots; the manner of creation of the set of LLM prompts to evaluate, leading to the generation of the hints; the manner to evaluate the resulting hint quality using human experts; and the evaluation of the attributes of interest of these hints. The overall method is shown in Figure 1 and explained further in the following subsections.

3.1 Student Snapshots

In order to provide example hints for ranking by educators, we needed some Snapshots of potential stuck students. We wanted these to be authentic so we decided to use a dataset of real students rather than construct synthetic examples. We chose to use the Blackbox dataset, which comes from users of the BlueJ IDE. These users are most commonly 16–18 years old [9] in secondary education.

To find Snapshots (stuck student states) for our study, we randomly sampled sessions from the Blackbox dataset [10] from an arbitrary week-day late in the typical northern hemisphere first semester (21st November 2023). One researcher manually selected states where they inferred that the students were stuck and in need of a hint, as evidenced by making no productive progress for some time after that point. Additionally, given the purpose was to use comparative judgement, states were selected that could be primarily judged from the context of a single method alone. Previous research has found that educators disagree about when is the best time to intervene [37, 56], so in lieu of clear recommendations from the literature, we used our own judgement. The exact point chosen is not crucial to the study, as long as it provides an interesting case for which to generate hints. For the same reason, the exact demographics or task of the students are not critical to the study as long as it is a useful test bed for generating hints – the hints are the primary focus of the study. As described later in subsection 4.6, multiple participants commented on the choice of student code being good, and realistic.

An initial candidate set of 38 Snapshots was extracted from the dataset on the given date, which was considered to reach saturation in terms of the variety of stuck student states. These Snapshots were filtered down to a set of 24 such that they were 8 sets of different types of Snapshots, with 3

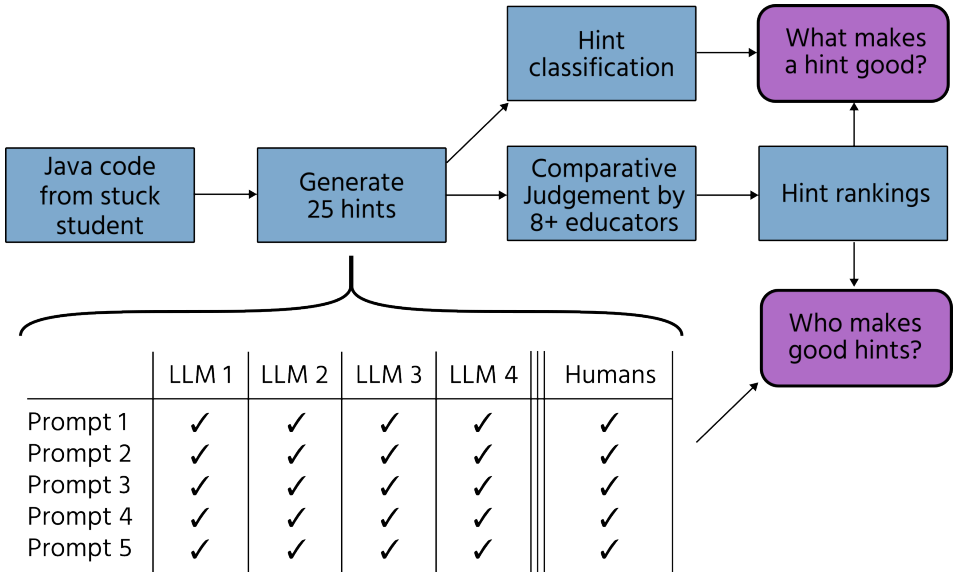


Fig. 1. The design of the study: We take a Snapshot of student code from Blackbox, generate 25 hints for this Snapshot (four LLMs each with five prompts, plus a hint from each of five human experts), and then ask eight or more expert educators to compare the hints to form a ranking. **We repeat this whole process with four different Snapshots and different educators, resulting in 100 hints ranked by more than 32 educators.** This allows us to infer what characteristics make a good hint, and which Generator (LLM+prompt, or human) generates the best hints.

in each set. This was then split into 3 sets of 8, which were intended to be used as training sets 1 and 2 (i.e. for prompt construction), and the test set (i.e. to use for the real experiment).

One example of a Snapshot is shown later in the paper in Figure 7 on page 23.

3.2 LLM choice

We deliberately chose to use a variety of LLMs in this study so that we could examine the variation between LLMs, and see the effects of the same prompts using in different LLMs.

We chose four LLMs: Mixtral-8x7B, GPT-3.5, GPT-4 and Gemini. These were chosen in April 2024. At the time GPT was by far the most popular LLM which justified picking two versions: GPT-4 was very new and GPT-3.5 was in very common use. Gemini was also newly released and provided contrast to GPT as a high-powered LLM. Mixtral-8x7B was a different kind of model that was designed to be run locally (on suitable hardware), so provided a contrast again to the other two models.

3.3 Prompt formation

Current literature does not yet suggest potential successful prompt templates for hint generation. Furthermore, as AI models evolve, which prompts are effective may change over time. To try to future-proof our methodology as much as possible when models continue to evolve, we used the following approach. Five researchers, who are also experienced programming educators with significant teaching experience, experimented with four LLMs (see previous section) to formulate prompts. Each model was used by a different researcher, but with five researchers and four models, we decided that the most recent model at the time, GPT-4, would be used by two researchers.

In order to formulate prompts, a set of Snapshots was extracted (see subsection 3.1) for the researchers to use to generate candidate prompts. Ultimately we decided that a single "training set" for prompt construction was best to allow better comparison of prompts with previous efforts during refinement. The crucial detail is that the set of problems used to construct the prompts was different (but stratified to have similar problems) than the set on which the final prompts were used to generate hints to show our educator participants.

As suggested by research into brainstorming [84], the researchers first brainstormed multiple prompts individually by using the sample Snapshots. Then the researchers all met together. Similar prompts were combined and refined experimentally, during two iterative cycles of collaborative discussion, resulting in a final set of five distinct prompts. The researchers decided jointly in the meeting which prompts were producing the best hints, but there was also a deliberate effort to construct a set of the best distinct prompts, rather than retaining small variations on the same core prompt.

3.4 Human-generated hints

Alongside the AI-generated hints, we also supplied human-generated hints. Each of the five researchers on our team was tasked with constructing one hint for each Snapshot in the experiment. This was an attempt to produce the optimal hint that they, as experienced educators, could provide to the student at that point, and it provided us with a benchmark in our results: we can not only compare automatically generated hints against each other, but compare them to hints produced by experienced humans. Constructing these hints was done independently; no researchers saw another researcher's hints until they had completed writing their own. None of the hints were modified or refined thereafter; they were used in the experiment just as the researcher had originally written them. In this way, it replicates a teaching scenario where an instructor would give a hint to a student with no external assistance or feedback on whether it was a good hint. Given that the hints are produced by independent individuals, we feel that the five researchers are an equivalent choice to any other educator we could have asked. All of the researchers felt motivated to show that they could produce a better hint than the LLMs.

3.5 Educator evaluation

We chose to evaluate the hints by asking Java educators to rank them. We could have asked students, but we believed that students would lack the capacity and metacognition to be able to read someone else's code, fully comprehend why it was faulty and understand the possible ways to fix it, then read multiple hints suggesting some of these ways to fix it, and rank which was best. We felt that only educators would be capable of such a task, since this aligns very well with their standard practice (read student's code, understand why it's wrong, think of how to fix it, then construct a hint to give to the student on how to proceed).

There is a possibility that even educators may not be experts at deciding the best hints for students. There is evidence from multiple disciplines that teachers [32, 67] are inaccurate at predicting which tasks students struggle with, as well as evidence in programming that educators struggle to rank which errors are the most frequent [8]. It is possible that educators are also imperfect judges of which hints are best for students. However, we felt that they were still the best available evaluators.

We wanted to gather educators' own, potentially diverse, opinions on hint quality, rather than using them to process the hints using a single rubric (as was done, for example, by Roest et al. [86]). We felt it was not clear from prior work what a detailed rubric for hints would be (especially one that would produce a single ranking for hints, as opposed to judging on multiple attributes like readability but with no clear way to move from that to overall quality), so we explicitly chose to use the educators' own judgement based on their experience. We chose a single unnamed dimension of

"best" rather than, for example, relevance, usefulness and learning that was used by Paaßen et al. [71]. Ultimately we want to find out which hint to show, and it was not clear how we should weigh such dimensions so we effectively transferred that weighing to our educators and asked them for a single dimension.

Thus we chose not to provide a detailed rubric or guideline for judging what makes the best hint. We decided it was necessary to provide at least a rough age range for the target students (a hint for primary school children would be quite different to university students!) so we chose the age range of the typical students in our Blackbox dataset from which we sourced the Snapshots. Our instruction to educators was therefore: "Imagine that you are helping a student who is somewhere within their first year of programming instruction, around the ages of 16-18. They are working on a problem and have become stuck and asked for help. Imagine that the computer they are working on could give them an automatic hint at this stage. We want you to determine which is the best hint to give in each circumstance."

3.6 Ranking hints: comparative judgement

One way to rank hints is to ask educators to rate each one on an absolute scale, say 1–10. It can be difficult to evaluate how good a hint is on such an absolute scale. Is a particular hint a 5/10 or a 6/10? Is everything just 7/10? Can participants remain internally consistent, and can the scale be consistent between participants?

An alternative method to produce a ranking is a technique called *comparative judgement*. The key idea behind comparative judgement is that people (termed judges) produce more reliable judgements when repeatedly asked to compare two items and pick the best one, than to rate each one individually on an absolute scale. Instead of "rate this hint 1–10", the problem becomes "which of these two hints is better". By asking judges to pick the best item from a set of random pairs, the judges act like the comparison function in a bubble sort, to sort the hints into an ordered list from best to worst. This form of judgement is more consistent across different judges, as it does not rely on an abstract absolute scale that would be influenced by different standards of individual participants.

Comparative judgement has been improved upon in an algorithm known as adaptive comparative judgement which minimises the number of comparisons needed [74]. Intuitively, if one hint is chosen several times as always worse than others, you can leave it near the bottom of the list and focus on the more borderline comparisons, to sort the list with fewer comparisons. Adaptive comparative judgement has been widely used for assessment in education and found to be reliable [5], and has been used in other areas of education research [38].

Each educator (judge) is first shown the Snapshot, and then asked to repeatedly rate which of two presented hints are better in this circumstance, with the pairs generated by an adaptive comparative judgement algorithm. San Verhavert and Maeyer [87] performed a meta-analysis on non-adaptive comparative judgement and found that the number of judges did not impact reliability, so we will not specify a minimum sample size of judges. If the judges are experts, San Verhavert and Maeyer [87] found that for 90% reliability, 26 to 37 presentations per item are needed. Since each comparison is between two items, ranking of N items requires $13N$ judgements to achieve 26 presentations of each item. Given that adaptive comparative judgement aims to reduce the number of comparisons, fewer should be needed. We use the No More Marking [48] platform to present the hints and collect the judges' choices. This platform uses a Progressive Adaptive Comparative

Judgement algorithm[1] and recommends 10 comparisons per item[2]. In our study, we ranked 25 hints for each Snapshot, so we required ≈ 250 comparisons, which we split across 8 judges doing 31 comparisons each.

For our experiment, we needed the expert educators who evaluated the hint quality to familiarise themselves with the Snapshot for which the hints were evaluated. To reduce the overhead incurred by educators, we chose to expose each participant to only one Snapshot. In order to help the participants all understand the Snapshot (which we took to be a prerequisite of the task of judging the hints, rather than something left to the educators to succeed or fail at) we provided brief notes with our interpretation of what the problem with the student code was. These notes were not given to the LLMs when generating the hints.

3.7 Measuring characteristics of hints

To characterise the best hints, we measure the following attributes:

- length of the hint (in number of words),
- complexity of the vocabulary (i.e. reading level),
- sentiment,
- correctness, and
- type of feedback (as per Cucuiat and Waite [16]).

The first two attributes are suggested in guidelines by Denny et al. [20] as used by Prather et al. [76] for error message readability.

In choosing a feedback framework to use, we could have chosen Cucuiat and Waite [16] and their work inspired by feedback literacy, or the work by Keuning et al. [42] inspired by previous work by Narciss [66]. It is worth discussing that choice here.

Several of the feedback components of Keuning et al. [42] do not apply in our context, for example:

- "Knowledge of performance for a set of tasks (KP)": summative feedback – outside our scope.
- "Knowledge of result/response (KR)": whether the solution is correct or incorrect – all of our solutions are incorrect.
- "Hint on task requirements (TR)": whether to use certain techniques – our hints are deliberately unaware of the task.
- "Test failures (TF)": whether a program failed tests – our Snapshots have no tests.
- "Style issues (SI)": whether a program has style issues – this is not part of our next-step hints for solving the problem at hand.
- "Performance issues (PI)": for example, taking too long to run – this was not applicable to our choice of Snapshots.

The feedback categories of Keuning et al. [42] that we believe are relevant to the present study are:

- "Knowledge of the correct results (KCR)": a description of a correct solution.
- "Explanations on subject matter (EXP)": explanation of a general principle.
- "Compiler errors (CE)", "Solution errors (SE)": errors from the compiler or runtime or logical errors. In our context, these are all effectively collapsed, because the hints just discuss errors in general, whether they would be compile-time or logical.

[1]See https://blog.nomoremarking.com/progressive-adaptive-comparative-judgement-dd4bb2523ffe, visited 4 November 2024.
[2]See https://blog.nomoremarking.com/using-comparative-judgement-in-different-subjects-at-ks3-4415195f8947, visited 4 November 2024

- "Bug-related hints for error correction (EC).": feedback that "clearly focuses on what the student should do to correct a mistake".
- "Improvements (IM)": suggestions for improving the structure, style or performance of a solution.
- "Knowledge About Meta-cognition (KMC)": "deals with a student knowing which strategy to use to solve a problem..."

The feedback categories of Cucuiat and Waite [16] are:

- "Telling": instructions on exactly what to do next.
- "Guiding": explicit feedback about the work above and beyond just direct accompaniment of what to change.
- "Developing understanding": explains the more general concept.
- "Opening up a different perspective": developing a higher-level understanding and considering alternative approaches.

There is no clear one-to-one mapping between these categorisations, but we consider that telling relates to the KCR and EC categories, guiding relates to the CE and SE categories, developing understanding to the EXP category, and opening up a different perspective to the IM and KMC categories. We chose to use the more parsimonious categorisation of Cucuiat and Waite [16] rather than a subset of the larger Keuning et al. [42] categorisation because we believed it was more likely to apply to our quite limited and specific context of generating hints, rather than the work of Keuning et al. [42] which covered a wide variety of literature and techniques for feedback in programming.

3.8 Pre-registration and ethical approval

The study was approved according to the ethics procedures of King's College London, approval number MRA-23/24-41449.

We pre-registered the design of this study on the Open Science Foundation (OSF) website – see https://osf.io/x8u3t/. The main changes since pre-registration are:

- We decided to show each educator-participant only one Snapshot rather than several, to reduce the load on each participant.
- We refined the exact process that we used to generate the hints, as described in subsection 3.3 and had the idea to add human-generated hints.
- We chose a new way to characterise hints using feedback literacy after reading the paper by Cucuiat and Waite [16] that was published after we pre-registered.

4 Results

The study was carried out in 2024, with the hints generated in April 2024 and educators recruited to perform the comparative judgement in August and September 2024.

4.1 Open data

All of the materials and data from this study are available publicly in the supplementary material for this paper, and also in an OSF repository: https://osf.io/p436s/ including: all of the stages of prompt creation and merging, all of the Snapshots, all of the generated hints, all of the processing performed on the hints, our participant instructions, our survey design, all of our survey results and all of the results of the hint comparison, plus the full processing pipeline for all of our statistical analysis and figures. We hope that this is useful for anyone interested in verifying or replicating our work.

4.2 Prompt creation

The five LLM prompts, shown in Table 1, were created as described in subsection 3.3. Two of these are multi-stage prompts, which involves asking the LLM for an answer and then feeding it back to the LLM. This may seem odd to those unfamiliar with using LLMs, but LLMs are not idempotent: when asked for information or an answer, and subsequently asked to use or improve it, – even without giving any new information in the second request – an LLM will generate a different and potentially improved answer.

4.3 Hint generation

The prompts were fed to the models in April 2024 to generate the hints. We manually unified the formatting between the resulting hints, to make sure the code snippets were all highlighted in the same manner regardless of which model generated them. We also made one minimal pass to remove boilerplate greetings from the very start or end of the hint, but we did not process the hint any further, to retain authenticity of machine generation. Examples of things we *did* remove:

- Phrases responding to our prompt request such as "Certainly!" or "Absolutely!", or in the case of multi-stage prompts: "Here's an improved version of my initial response".
- Leading salutations such as "Hey there" or "Alright, student!"[3].
- Trailing salutations such as "Best regards." or final remarks such as "If you need anything else, just ask."

We felt that all of these phrases could be removed automatically in future by refining the prompt or using a second processing pass, and they distracted from the hint content we wanted to evaluate.

Examples of things we *did not* remove:

- Encouraging phrases such as "Good luck!" or "Keep it up."
- Emoji, e.g. an airplane emoji in a response about a student stuck calculating air miles points.
- Use of first-person phrases, e.g. "I noticed that there are a few things worth looking into".
- Cases where the AI has addressed the educator such as "Now, to assist your student further, I recommend..."

The final item is essentially the prompt "misfiring" and we felt it was important to penalise the prompt and model for this behaviour.

All of the exact changes made to the generated versus presented hints can be seen in our OSF repository and supplementary material.

The final set of hints included 25 per Snapshot: four AI models (Mixtral-8x7B, Gemini, GPT-3.5, GPT-4) with all combinations of the 5 constructed prompts (see Table 1), plus five human-generated hints. All hints were formatted similarly and given arbitrary identifiers to obscure how they might have been generated.

4.4 Judges

We recruited participants to act as judges via the SIGCSE-members and a US high school teacher mailing lists, forums for school teachers in the UK and for teacher-users of the Java programming systems BlueJ and Greenfoot, and on the researchers' personal social media accounts. We asked for "Java educators" to complete an online task plus survey. We did not screen participants for eligibility but based on the survey results they were in general very experienced. There was no reward for participation. 85 participants signed up; three started but did not finish the task, 41 completed the task, and 35 of those completed the survey (we discuss these numbers further in the threats to validity in section 7).

[3]One of the authors plans to use the latter to communicate with their students in future.

#	Prompt
1	I'm learning to program. Without giving the solution, can you guide me into the next step to fix the problem in this code? $CODE
2	You are an experienced programming instructor teaching a course in Java. I will provide you student code and your task is to generate a hint for the student. Please do not provide the full solution, but try to generate a hint that would allow the student to proceed in the task. Please address your response to the student. Here is the code: $CODE
3a	You are an experienced programming instructor teaching a course in Java. I will provide you student code and your task is to try infer what the task the student is working on is. Here is the code: $CODE
3b	You are an experienced programming instructor teaching a course in Java. I will provide you student code and an explanation of the task they are working on, and your task is to generate a hint for the student. Please do not provide the full solution, but try to generate a hint that would allow the student to proceed in the task. Please address your response to the student. Explanation of the task: $PREVIOUS_ANSWER Here is the code: $CODE
4	You are a tutor who is helping a student who is stuck while writing code for an introductory programming course. To help the student, you must generate a hint that will identify their mistake and help them make progress continuing with their code. If there is a serious error, please point that out first. Please be helpful and encouraging, but do not reveal the answer - instead, make one concrete suggestion to help them progress. Here is their code: $CODE Please identify the most serious error in this code and suggest a hint to the student to help them. Just show the "Hint", suitable for immediate display to the student. Please do not include any other explanatory text.
5a	I have written the following code: $CODE I need help. Act like a good teacher and give me some help. Respond in no more than three sentences.
5b	Do you think the hint you just gave was a good hint? Critique it, including consideration of accuracy, friendly tone, helping the student to learn and not giving too much of the answer away.
5c	In light of your criticism, improve the response you first gave.

Table 1. The five prompts given to the AI to generate the hints. The Snapshot is substituted wherever "$CODE" appears. Prompts are separated by double-lines. Some (3a+3b, 5a+5b+5c) are multi-stage prompts, separated by a single line; each prompt part is given in sequential order; the marker "$PREVIOUS_ANSWER" indicates that the full answer to the preceding part of the prompt should be inserted.

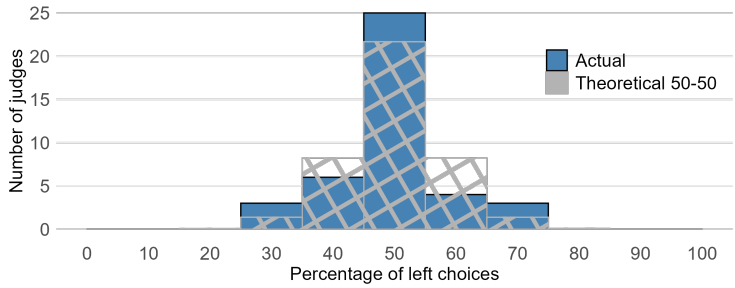


Fig. 2. A frequency histogram based on how many participants made a given percentage of "left" decision when asked to make a left-vs-right judgement. The blue bars show the actual values; the grey hashed bar overlay is the theoretical distribution if the decisions were completely random (i.e. drawn from a binomial distribution with probability of 50% for each decision).

Participants were assigned in a round-robin fashion to try to ensure that as many Snapshots as possible had the necessary 8 completions, which proved difficult with the low completion rate. Ultimately, four Snapshots reached the required 8 completions (with 8, 10, 8, 9 completions). Two more Snapshots had only three completions each so are excluded from all the analysis of the comparative judgements and hint rankings – but survey results of the judges of these Snapshots are retained for the purpose of analysing judges' reflections on performing the task.

4.5 Validity of Judging

Given that our results are reliant on the comparative judgement task, it is important to check that the participants took the task seriously. We employed several metrics for this purpose.

The first check was how often the participants selected the left option as the best hint when given the left-vs-right decision to pick the best hint of a presented pair. If any/many participants consistently just clicked the same button it could indicate boredom during the task. Figure 2 shows this data, compared to the theoretical binomial distribution that should occur if the percentage was 50-50%. A chi-squared test ($\chi^2 = 44.0, p = 0.06$) between the two was not significant, suggesting the observed data was not significantly different from a 50-50 distribution of left-right clicks – informally, this suggests there was no statistically significant bias towards repeatedly clicking the left or right button.

The second check was how long participants took to make the judgements, to see if this suddenly dropped off during the judging, which would again suggest boredom during the task. Figure 3 shows these results. The participants take a long time for their first (≈75 seconds) and second (≈50 seconds) judgements as they acclimatise to the task, and this is followed by a gradual speeding up (from around 40 seconds to around 20 seconds) as they work through the task. This gradual pattern suggests becoming more accomplished at the task rather than giving up.

Those results suggest that judges took the task seriously, but we must also examine a core assumption of comparative judgement: that there is an underlying ranking of items that is shared among the judges. In a perfect case, such as a task asking judges to pick the largest number, it should be possible (excepting human error) to perform a perfect ordering. Most real-world tasks, however, will have some expected disagreements just because people have differing opinions. This is acceptable as long as the disagreements are not too wide-scale. To investigate this, two measures for inter-rater reliability in comparative judgement may be used: Reliability (a per-task measure) and Infit (a per-judge or per-item measure).

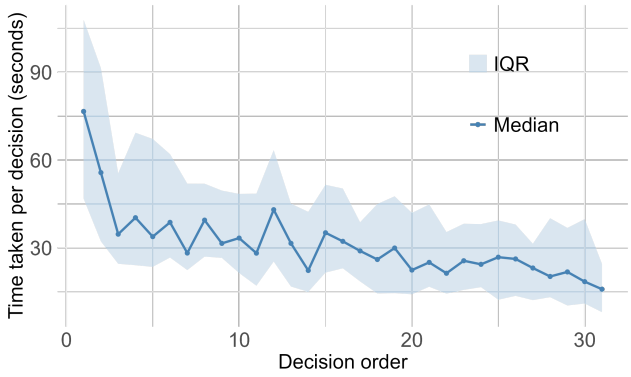


Fig. 3. The median (line) and interquartile range (shaded area) of time taken to make each judgement, ordered by the decision order (first judgement made on the left, through to the last on the right).

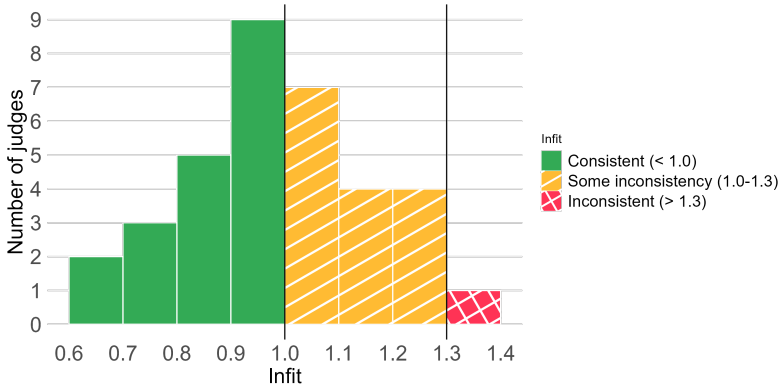


Fig. 4. The frequencies of different Infit ratings, one rating per participant.

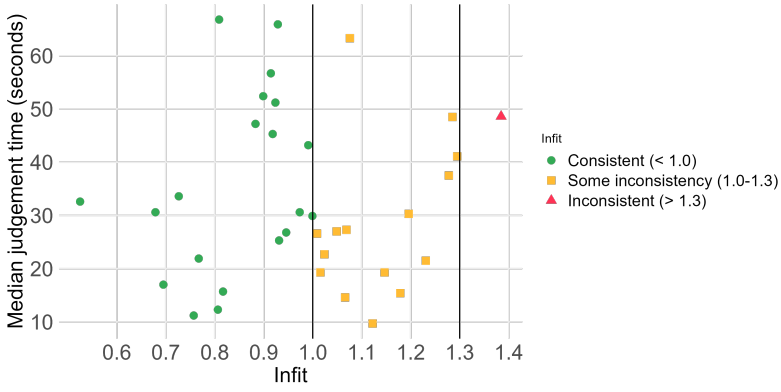


Fig. 5. A plot of Infit ratings (each plotted point is a participant) against median judgement time, to see if less consistent judges were judging faster.

Our Reliability scores for the four Snapshots were 0.73, 0.76, 0.76 and 0.60. A reliability of 0.7 is generally considered sufficient [87] so the majority of our Snapshots showed outcomes with a high level of inter-rater consistency. Our Infit scores are shown in Figure 4. Note that inconsistency in Infit scores refers to *inter*-judge consistency not *intra*-judge: it is about whether the judge agrees with their peers, not whether their own decisions were self-consistent. An inconsistent judge in this sense does not necessarily mean a "wrong" or "bad" judge, just one whose opinions differ from the other judges. We therefore did not exclude these judges. We did check if inconsistency was associated with faster judgements (suggesting a kind of speed-accuracy trade-off; maybe the inconsistent judges were choosing arbitrarily in a rush) but this was not the case (see Figure 5, confirmed by a linear regression being non-significant, $p = 0.693$).

Given that several previous studies of educators [8, 37, 56] have found that educators do not always reach good agreement on pedagogical issues, it was slightly surprising that the agreement level was so high.

Finally, we checked the speed of the hint judging against reading speed. A statistical model (linear regression of log-transformed time taken vs combined word count and participant, $p < 0.001$ for the combined word count factor) found that the average time taken to choose between two hints by combined word count was as follows:

- 100 words: 28 seconds
- 200 words: 31 seconds
- 300 words: 36 seconds
- 400 words: 41 seconds
- 500 words: 46 seconds

Given that the average reading speed for non-fiction is 238 words per minute [11], it is likely that participants were not fully reading most of the hints. However, this does not mean the judging was invalid, for several reasons:

- It is valid for a participant to immediately assess that a hint is too long and that students will not have the patience to read it, without reading it themselves.
- Participants saw the same hints multiple times, so they may have begun to recognise hints without reading them through a second time.
- The participants are likely highly educated, and therefore used to skimming complex text effectively.

In summary, all our checks suggest that the judges took the task seriously, and that there is good if imperfect agreement among educators as to which hints are good.

4.6 Method and stimuli evaluation

In our survey we asked participants whether they found the comparative judgement task easy or hard via a free-text response. 18 of 35 said they found it easy, 7 indicated a medium difficulty, 8 indicated they found it hard. Only one participant mentioned boredom.

We asked participants for their observations about the Snapshot example that they saw, in a free-text response. Not all participants gave a response. Five mentioned that the Snapshots were well-chosen, eight said they thought they were realistic, and two said they thought they were unrealistic or outliers.

4.7 Hint characteristics

To evaluate which characteristics of hints were associated with their ranking, we extracted various attributes of the hints, shown in Table 2.

Attribute	Type	**Method of extraction**
Word count	Integer	Count number of words, including words in code, but ignoring punctuation, symbols and emoji.
Reading level	Real	Flesch-Kincaid Reading Grade Level [45].
Sentiment	Real	VADER Sentiment Analyzer [35], compound polarity score.
Model	Category	The name of the AI model or the term "Human" to indicate how the hint was generated.
Telling	Boolean	Feedback-literacy-inspired category. See subsection 4.7.
Guiding	Boolean	Feedback-literacy-inspired category. See subsection 4.7.
Developing understanding	Boolean	Feedback-literacy-inspired category. See subsection 4.7.
Opening up a new perspective	Boolean	Feedback-literacy-inspired category. See subsection 4.7.
Partially incorrect	Boolean	Flag indicating whether a hint was partially incorrect.

Table 2. All of the hint attributes that were examined, and how the attributes were extracted.

The word count is straightforward, while the sentiment analysis [35] and reading level [45] used existing techniques. We did not initially plan to use Model as a factor given that the participants should be unaware of which model was used. However, since the Mixtral-8x7B model generated longer and less readable hints in a particular style, we were unsure if an effect of word count would be related to the word count, the readability, or the model's individual style, so we included Model as a factor to evaluate if it had an effect not captured by the other factors.

All of the other attributes were categorised by the researchers. The "Partially incorrect" flag is a relatively unambiguous technical check for incorrect parts of the hints (if any part was incorrect this flag was true, even if the rest was correct). Consider, for example, this [erroneous] student code from one of our Snapshots in the study:

```
String letter=sc.nextLine();
if (letter.equals(a|| c|| e ||g)))
```

Multiple hints suggested a fix that included code similar to the following:

```
if (letter.equals('a') || letter.equals('c') ...
```

However, because `letter` is a String and `'a'` is a character, they will never be equal in Java even if `letter` has the value `"a"`, because the types (String and Character) do not match. Most of the incorrectness was subtle in this way, but we included it as a factor to see if it affected the relevant hints' placing.

The final four factors in Table 2 are inspired by feedback literacy theory, introduced by McLean et al. [64] and made known to us by its use in the study of programming error messages by Cucuiat and Waite [16]. We adapted the four themes into categories of the same name by creating a set of definitions that applied the concepts to next-step hints, as follows. Two researchers initially tagged some hints which were unused for the study, in order to calibrate. Then they both categorised all 100 hints that were judged by educators for the four completed Snapshots. All hints were tagged as yes/no for the four dimensions, giving $2^4 = 16$ possible overall categories. In their initial independent tagging the researchers reached an agreement of 65%. The disputes were resolved in a meeting between the two researchers, and primarily revolved around clarifying the definition

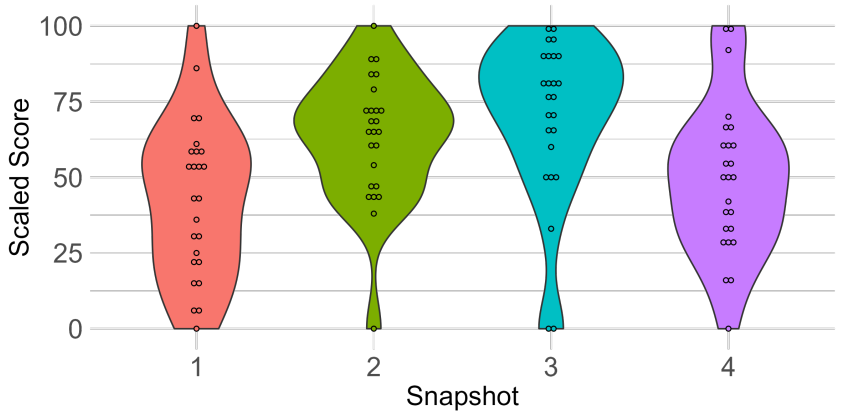


Fig. 6. The scaled scores of the hints, split by Snapshot on the X-axis. Each dot is a hint, and the violin plot shows the same data as the hint dots, but is added to aid visualisation. Snapshot 2 (and to a lesser extent Snapshot 3) have clear outliers at the bottom of the graph.

of the "Developing understanding" tag. The resulting definitions after clarification are shown in Table 3.

The frequencies of the different combinations of concepts are shown in Table 4. More than half the hints contain "Guiding" with no other concepts. With the hints being so similar in this regard, it will naturally reduce the discriminability that these feedback categories can provide.

4.8 Hint scoring

To arrive at a ranking, all Generators are applied across all Snapshots, and hints are scored for each Snapshot. Comparative judgement, as implemented in the NoMoreMarking platform that we used, supports two different ways of scoring hints. One is a simple ranking: best hint, second-best, and so on. The other is a "scaled score" which estimates how far apart the items are on a normalised scale (0 being the worst item, 100 being the best). We had intended to use the latter as it was more informative. However when we looked at the data we realised a problem. As Figure 6 shows, in the case of Snapshot 2 (and to a lesser extent 3), one hint (or three hints) are so poor that the rest of the hints have their scores pushed up the scale. Therefore a weak performance on Snapshot 2 is penalised less than on Snapshots 1 or 4 when we collapse a Generator's performance across Snapshots. This would artificially skew the results, and thus we opted against using the scaled scoring.

This problem is to some extent present even in the hint rankings. Since we have no cross-Snapshot normalisation (each participant only judged one Snapshot) we cannot determine whether, for example, all the hints on Snapshot 1 are better than all the hints on Snapshot 2. But since each Generator provided a hint in all contexts, it at least counter-balances across the Snapshots. We felt that using the ranks was a better choice than scaled score to minimise the effect of this warping within Snapshot.

4.9 Hint rankings

Based on the reasoning in the previous section, we thus used the ranks of the hints within their Snapshot as our dependent variable. The ranks were mapped to ascending scores to be more intuitive, so that a rank score of 25 was the best hint for a Snapshot, 24 the second-best, down to

Concept	Description
Telling	The hint contains instructions on exactly what to change in the code. This could be actual code (e.g. “insert x = 0;”) or words that reach the same outcome (e.g. “You need to assign 0 to x before the loop” or “you should start the loop at 0 not 1”). The feedback requires no or little extra thought from the student other than to follow the instruction. The instruction is either exact (delete this line) or close-to-exact (move this line to after the loop). It does NOT include feedback that requires more thought or has ambiguity in exactly what needs doing (e.g. “You need to update the loop variable somewhere within the loop” or "The function call should not be inside the loop").
Guiding	The hint contains explicit feedback about the original code above and beyond just direct accompaniment of what to change. So if it says “your loop is incorrect; you should start from 0” this is only telling, not guiding. This might include statements like “your loop will run forever” or “you are not updating x anywhere after its declaration” or “consider whether the loop will terminate” – provided it does not also include an ensuing exact instruction on the fix (which would be categorised just as telling). It can include positive specific feedback like “Your loops are the correct structure”. It does NOT include *generic* feedback like “you’re so close” or “this is a great start” which could be applied to any piece of code.
Developing understanding	The hint contains a more general explanation of a concept or of the needed change to the code. For example, it might explain why List cannot be indexed with square brackets in Java. It may also point to places where students could find out more information for themselves (e.g. a URL or what concept to research). The key is that it is explaining the general rule which would also apply to future coding, not just the specific issue with the current code (which would only be guiding). This may be phrased as a general tip or tip-for-the-future, the key is that it is more general than just this specific code example.
Opening up a different perspective	The hint contains a suggestion of a different approach to solving the problem (e.g. using a find method rather than manually looping through the list to search, or using a different programming language altogether). This does NOT include cases where the student has not even made a coherent start; it needs to be in contrast to the student’s current thinking or approach to the problem. If they have done nothing, the suggestion might be telling or guiding depending how specific it is.

Table 3. The complete definitions of the four concepts derived from feedback literacy theory [64] that were used to manually characterise the hints.

Telling	Guiding	Developing	OpeningUp	Frequency
	✓			56
	✓	✓		17
✓	✓			7
✓				6
✓		✓		3
✓	✓	✓	✓	3
✓			✓	2
	✓		✓	2
✓		✓	✓	2
		✓		1
✓	✓	✓		1
				0
			✓	0
✓	✓		✓	0
		✓	✓	0
	✓	✓	✓	0

Table 4. The frequencies of all possible different combinations of the four feedback literacy concepts (see Table 3) in the 100 hints, ordered by their frequency. ✓indicates the presence of the concept.

1 for the worst hint. Thanks in part to comparative judgement being a forced-choice paradigm, there were no ties. The scores are thus "zero sum"; there are 4 scores of 1 (one for each Snapshot), 4 scores of 2, etc, up to 4 scores of 25. The scores are all relative: if one hint is better, the others must be correspondingly worse. The midpoint (both mean and median) hint is thus 13 by definition, and what is of interest is which factors lead to higher placings. This metric is termed *RankScore* and is the way we compare the quality of the hints for the remainder of the paper.

The rankings of the specific hints by themselves are not of direct interest in this paper (although the full set of hints and their ranks can be seen in our OSF repository/supplementary material); the interest lies in the accompanying factors, such as which Generator produced the best hints, or which attributes (see subsection 4.7) were associated with high-ranking hints. To give a sense of the hints, Figure 7 shows the code and the best and worst hint for Snapshot 1.

4.10 Hint rankings by generation mechanism

Figure 8 shows the RankScore of the hints grouped by the Generator that produced them (with all five prompts collapsed into a single score for each LLM). As can be seen, GPT-4 has the highest mean rank, followed by humans, with Mixtral-8x7B the worst. A Kruskal-Wallis test confirms that the effect of Generator was significant ($\chi^2 = 11.711$, $df = 4$, $p = 0.020$) at the 5% threshold for statistical significance. To investigate the specific claim of interest as to whether LLMs beat humans, a Mann-Whitney U test of humans versus GPT-4 across all prompts was non-significant ($U = 250$, $p = 0.180$) but humans versus GPT-v4 with its best prompt (3) is significant (W = 67, p-value = 0.040). That is, GPT-4 with the best prompt outperforms humans in generating hints that are more highly ranked by human evaluators.

For the AI-generated hints, Figure 9 shows the rank scores split by the prompt used to generate them. The prompt numbers correspond to Table 1. Prompt 3 produces the best results, and prompt 4 produces the worst results.

Student code:

```
public static boolean isPalindrome(String input)
{
    int length = input.length()-1;
    String palindrome="";
    for (int i = length; i == 0; i--) {
        System.out.println(i);
        char letter = input.charAt(i);
        palindrome = palindrome + letter;
        System.out.println(i);
    }
    if (palindrome == input)
    return true;
    else
    return false;
}
```

Notes:
There are two core problems in this code. One is that the for-loop condition for continuing is wrong: it should be `i >= 0` rather than `i == 0`, which means the loop will not run at all for most inputs. The other problem, even if the loop did run, is that the code uses == to compare strings after the loop, rather than `.equals()`.

Best hint (GPT-4, Prompt 2):
There are a few things you might want to reconsider in your code.

1. Look at your for loop condition. You've set it to `i == 0`, which means the loop will only execute if `i` is equal to `0`. Is that what you intended?
2. You're comparing strings using == operator. In Java, == checks if two references point to the exact same object, not their equality in terms of content. You might want to use a different method for string comparison.
3. Also, consider the case sensitivity of your input string. Does your method handle both lower and upper case letters?

Keep these points in mind and try to revise your code. Good luck!

Worst hint (GPT-3.5, Prompt 5):
Your loop condition should be `i >= 0` to iterate over the characters correctly. Remember to use `.equals()` method instead of == for string comparison, as == checks for reference equality. Additionally, consider using `StringBuilder` for efficiency when building the palindrome string inside the loop. This will help you understand why string comparison doesn't work as expected and improve the performance of your code.

Fig. 7. The top shows the code from Snapshot 1 (which was reproduced exactly, including the student's original spacing and indentation), followed by the notes which we gave to participants to explain the problems with the code. Below that we show the best hint and the worst hint (and their source) as ranked by the participants in our study.

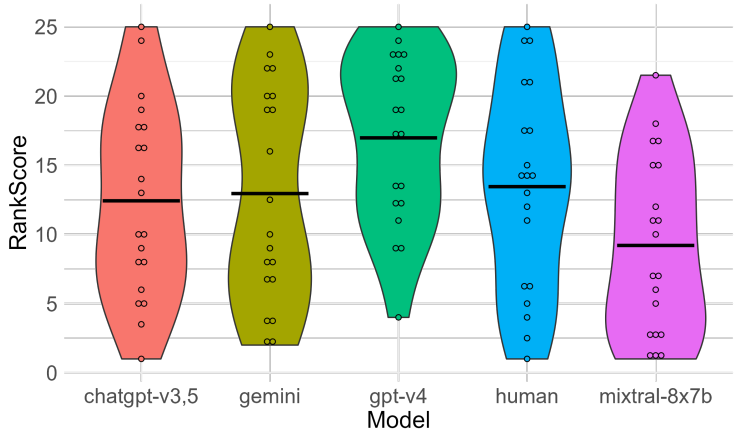


Fig. 8. The rank score of the hints (higher is better) split by the model that produced them, with different prompts averaged to a single value for each LLM. Each dot is a hint; the violin plot shows the same data as the hint dots and is added to aid visualisation. The black horizontal line is the mean rank of the hints generated by that model.

We have 100 hints overall: (5 AI models × 4 prompts + 5 human hints = 25 Generators) × 4 Snapshots. Thus for each AI model we have 20 data points (5 prompts × 4 Snapshots), for each prompt we similarly have 20 data points (5 models × 4 Snapshots) but for each Generator we only have 4 data points (one per Snapshot). Therefore we must be very cautious in interpreting this data because we may be interpreting noise; nevertheless it is shown in Figure 10. This seems to indicate there may be an interaction between prompt and model as to which is best. For example, although prompt 3 is generally best, it interacts very poorly with the Mixtral model. Similarly, GPT-4 is the best model but does poorly with prompt 4. The five humans who generated hints are shown in the same graph; it would appear that there is less variation between humans than between models or between prompts.

4.11 Hint rankings by hint characteristics

The analysis by hint characteristics is potentially completely orthogonal to the analysis of the hint generation mechanism. Here we are interested in attributes of the hints themselves regardless of how they were generated. We used the hint attributes described in subsection 4.7 to see what effect they had on the hints' RankScore.

In order to associate hint characteristics with their effect on the performance (i.e. ranking by educators) of the hint, we need some kind of model. Simple mathematical models such as linear regression assume a linear or at least monotonic relationship: for example, longer hints are worse or longer hints are better, but not a pattern in a U-shape or other non-linear pattern. However, it seems probable that attributes such as length or sentiment may have a more complex relationship than that. For modelling we turned to machine-learning classifier models, specifically decision trees and their improved variant, random forests [34]. The advantage of these methods over classical statistical methods is that they can identify complex non-monotonic patterns.

Decision trees are a classic data mining method where the data is used to create a binary tree that classifies the outcome variable, by splitting the values around a point (e.g. perhaps hints having a word count above 300 leads to a lower RankScore). The problem with decision trees is that they tend to overfit the data. Random forests solve this problem by extracting tens or hundreds of

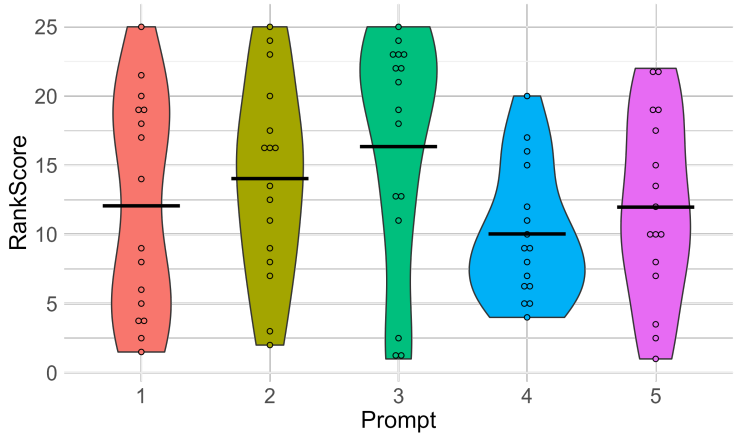


Fig. 9. The rank score of the AI-generated hints (higher is better) split by the prompt used to generate them. Each dot is a hint; the violin plot shows the same data as the hint dots and is added to aid visualisation. The black horizontal line is the mean rank of the hints generated by that model.

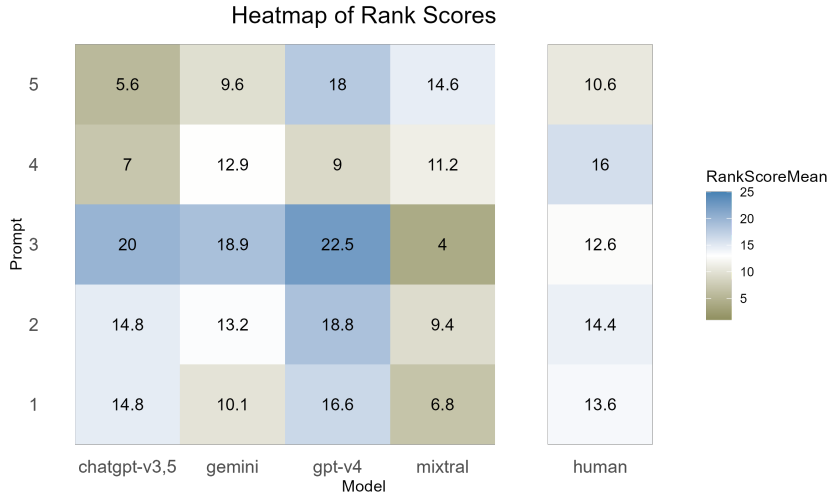


Fig. 10. A heatmap of AI model vs prompt (plus the five humans), showing the mean rank for that Generator. Each entry is derived from only four points (one per Snapshot) so it should be interpreted with caution.

random subsets of the data and available features, and then fitting a decision tree to each subset, yielding tens or hundreds of trees. The results of these trees are then averaged in a "forest" to form a classifier. This classifier could potentially be used to predict new rank scores based on a new hint, but here we are solely interested in introspecting which hint attributes were important for the classification of best hints and in what way (e.g. is higher better).

To perform the analysis we used the R ranger package [96]. Hyperparameters were tuned using a full cartesian search of plausible parameters (source code is available in our OSF repository/supplementary materials) with 5-fold cross validation (as suggested by [7]), using root mean-squared error (RMSE) as the target metric. The RMSE of the best model was 5.49, which means (with a

Input factor	Importance	p-value
WordCount	19.9	< 0.001
FleschKincaidGradeLevel	13.7	0.003
OpeningUp	3.2	0.006
Model	1.1	0.313
PartiallyIncorrect	0.5	0.207
Guiding	0.3	0.278
Sentiment	0.3	0.519
Telling	-0.1	0.587
DevelopingUnderstanding	-0.6	0.718

Table 5. The factors in the random forest model (outcome variable: RankScore), their importance (percentage increase in mean squared error of the outcome if omitted from the model), and p-value (calculated using the method of Altmann et al. [4]). Higher importance means the attribute was more important in predicting the RankScore of each hint. Zero or negative means that the factor was unimportant. A line is drawn to separate the three statistically significant values (i.e., $p < 0.05$) from the rest.

variance in our 1:25 ranks of 52) an equivalent R^2 is 0.42, suggesting that the model explains 42% of the variance. Although the outcome is interpretable in this model, it suggests that there are other factors beyond the hint characteristics that we have chosen that explain the rest of the variation in hint performance.

The main output of a random forest is the *importance* of each input attribute. Importance (technically, the percentage increase in mean squared error of the outcome when the input factor is omitted from the model, higher means the factor is more important, 0 or negative means totally unimportant) tells us which factors most influenced the outcome variable, although it does not indicate whether that influence caused the ranking to be higher or lower. The ranking by importance is given in Table 5. We calculate p-values using the method of Altmann et al. [4], and find that only the three most important features are statistically significant in their importance (at the $p < 0.05$ level).

By far the two most important factors were word count and reading level, followed by the feedback literacy item of "Opening up a different perspective". Note that Model was relatively unimportant, suggesting there are few lasting effects of how the hint was generated, once the other factors are taken into account.

To visualise the effect of the important attributes, in Figure 11 we graph partial dependency plots, which show the effect on RankScore for each value of the attribute. The dotted line across each shows the baseline, with each value of the hint attribute potentially increasing RankScore (better hint, above the dotted line) or decreasing it (worse hint, below the dotted line).

The results for word count reveal a "sweet spot" where hints that are 80–160 words long are ranked highly, around 4–5 places higher than hints with word counts below or above this range. Short hints are rated particularly poorly.

The results for reading level show that a lower grade reading level is better. The grade level scale here corresponds to grade levels in US schools, so for example a reading grade level of 9 (where the hint quality suffers a sudden drop) corresponds to 14 year-olds. So any hints not understandable by fourteen year olds are rated around 5 places lower than hints understandable by thirteen year olds (grade 8) and younger students.

The results for *opening up different perspectives* show that hints which suggest an alternative approach to the problem are ranked 2 places *lower* than hints which do not. Whereas *guiding* hints

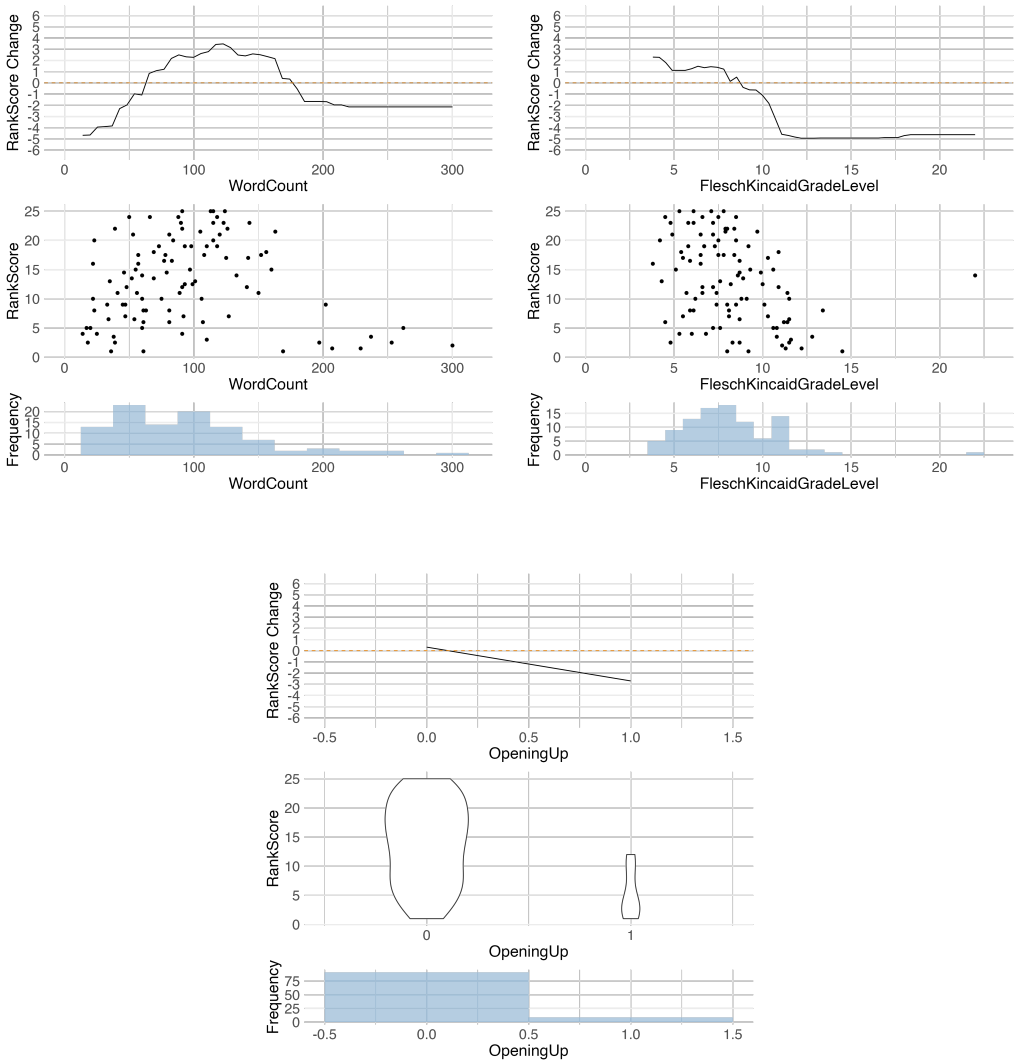


Fig. 11. Partial dependency plots for the three statistically significant factors in predicting which hints are best. Each plot is a vertical triumvirate: the top-plot shows the effect on the RankScore on the Y axis (above the orange line: better than average) by value of the attribute on the X axis as predicted by the fitted random forest model; the middle plot shows a scatter (or violin) plot of the actual data values from the 100 hints; and the bottom plot of each set is a histogram of the same data.

which offer additional guidance beyond exactly what to change are ranked around 2 places higher. See Table 3 for the definitions of these items.

5 Educator survey

As well as performing the comparative judgement task, we asked participants to complete a short survey in order to collect some information about themselves (specifically, their teaching

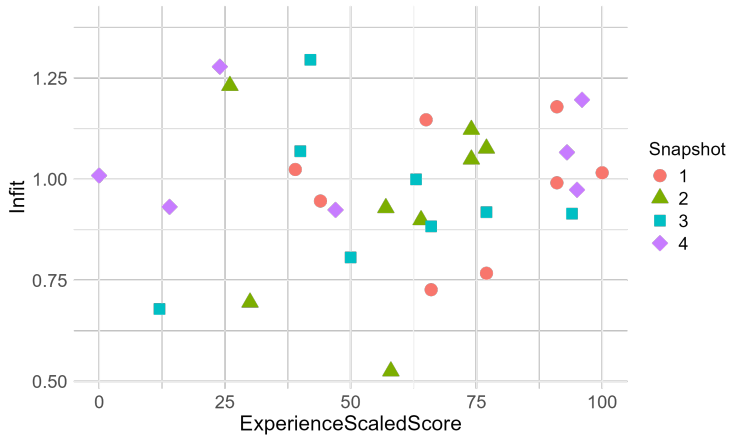


Fig. 12. A participant's experience (a scaled score ranked by researchers using comparative judgement) against their Infit (how much they agree with fellow participants: lower values of Infit indicate higher agreement with their peers).

experience), their opinions on the hints overall (which go beyond the rigid comparative judgement framework) and their experience of using comparative judgement to rate the hints.

We asked about their experience of teaching Java. Our past experience in other studies had suggested that a simple numeric field (e.g. "How many years have you taught Java?") was insufficient to capture the wealth and variety of experience. We asked them for a free text entry describing their experience of teaching Java. We then ranked these responses (using comparative judgement, but with the researchers as judges) on a loosely defined "Java educator experience" basis. This allowed us to sort the participants by experience and thus we can summarise their experience with an upper quartile, median and lower quartile:

- Upper quartile: "I have started teaching Java in 1998 and have 30+ years of teaching experience as a TA, scientific researcher, and educator. Most of it was done in Java."
- Median experience: "Teaching Java for more than twenty years, have taught Pascal, C, C++ before..."
- Lower quartile: "I have taught Java programming to High School students about 8 years. I also teach Scratch, HTML, Javascript..."

The comparative judgement also gave us a scaled score (as described earlier in subsection 4.8) from 0 to 100. We could then plot this against the (described earlier in subsection 4.5) Infit to see if there was a relationship between experience and agreement with peers, as shown in Figure 12. A linear regression confirmed there was no effect ($p = 0.60$) of experience on Infit.

One important survey question related to the overall opinions of the hints. Because our comparative judgement task is entirely relative, it cannot tell us whether all the hints were good or all the hints were bad, or somewhere inbetween. For this purpose we asked the participants how the hints compared to having no hint, and the results are shown in Figure 13 – the judges generally thought that most of the hints would be helpful. We asked the participants why they thought the hints were helpful (or not) as a free-text response, and performed a simple thematic analysis [13] to analyse these responses – the counts of different themes are given in Table 6.

Theme	Participant count
Help not hinder independent learning: Participants mentioned that hints should aid independent learning (e.g. by giving cause for the students to think) rather than hinder it (e.g. by providing the exact solution to the student with no need for further thought).	10
Context matters: Participants mentioned that they needed to understand more about the context of the student receiving the hint in order to decide whether the hint was appropriate or whether it needed further adjustment.	6
One at a time: Participants expressed a dislike for hints which addressed many errors at once, and stated they would prefer a hint which identified and focused on solving only one problem with the code.	6
Too long or complicated: Participants stated that some hints were too long or complicated to provide any benefit to a student, and expressed doubt that the students would read and/or understand such hints.	5

Table 6. The themes we identified in participants' responses about why the hints they saw would (or would not) be better than having no hint, plus the count of unique participants (out of 35) who mentioned this theme. The table is sorted by frequency.

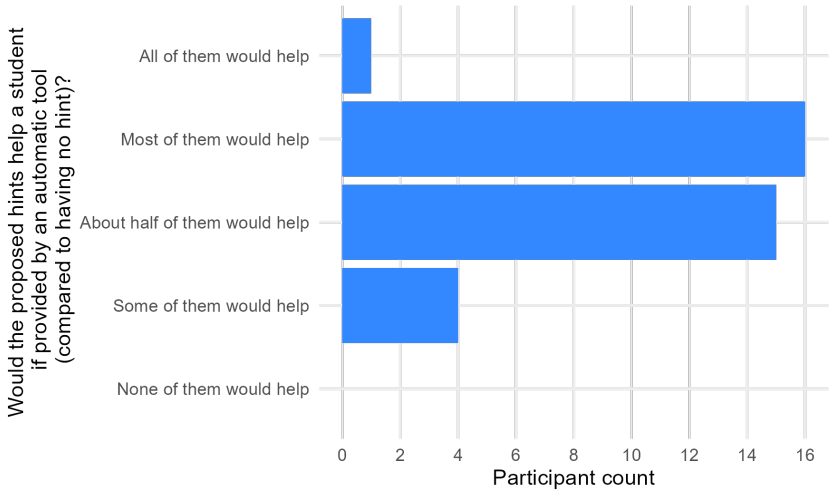


Fig. 13. Results of asking the participants whether the hints would better than having no hints on a 5-point Likert scale.

Participants could also offer their opinion on what they thought was important in a good hint, as a free text response. We similarly performed a simple thematic analysis to analyse these responses, and the counts of different themes are given in Table 7.

We also asked participants to state whether they felt they could do better themselves. The results are shown in Figure 14. It is important to interpret this finding in light of the experience result

Theme	Participant count
Conciseness: Participants preferred short, concise, to-the-point hints.	22
Hint not solution: Participants wanted hints that did not provide the exact solution, but rather a pointer or suggestion or thought-provocation that would involve the student thinking further.	19
Not over-praising: Participants disliked hints that over-emphasised praise or positive language.	11
Specificity: Participants preferred hints that were specific rather abstract and vague.	9
Correctness: Participants mentioned wanting correct, accurate hints (usually mentioned because they had spotted a hint they felt was incorrect).	9
Positive tone: Participants liked a positive or encouraging tone to the hint.	8
Not too short: Participants mentioned disliking very short hints as being unhelpful or lacking in useful detail.	5
Unhelpful summary: Participants mentioned disliking the tendency for hints to contain a summary of what the code was doing or trying to do, because they felt this was unhelpful.	4

Table 7. The themes we identified in participants' responses about which characteristics of hints were important to their ranking choices, plus the count of unique participants (out of 35) who mentioned this theme. The table is sorted by frequency.

described earlier in this section. Over half of our participants had the equivalent of 20+ years of Java teaching experience, and yet the vast majority of them felt their hints would be around the median hint in the study. This matches with our results which show that the human-generated hints from the researchers (several of whom would be in the top half of experience in the study) were around the median. We interpret that the hints in the study were generally considered high quality.

In a slight oversight on our part, we did not explicitly ask the participants whether they thought the hints were AI-generated. This was not part of our research questions but in retrospect it may have been useful to ask. Four participants spontaneously made reference to AI or LLMs in their responses; one said the hints "feel like more of what an AI might respond with", one said "One hint seemed to contain a bit of a [LLM] prompt.", one said "LLM / ChatGPT levels of positivity would be irritating over time", and one suggested that another research group "is also experimenting with AI-generated hints". We suspect that most participants inferred or assumed that the hints were AI-generated; the surprise for them might instead have been that some were human-generated, rather than all being AI-generated.

The full results of the survey are available in our OSF repository/supplementary materials.

6 Discussion

This study has findings in multiple dimensions, which we will discuss in turn.

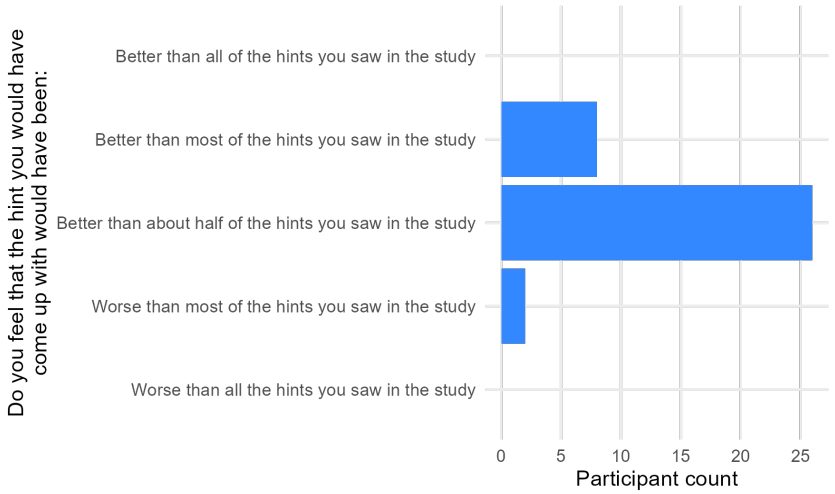


Fig. 14. Results of asking the participants whether they felt they could make better hints than those in the study, on a 5-point Likert scale.

6.1 Hint characteristics

We analysed the ranking of hints against their characteristics in order to investigate which characteristics of the hints were most important, according to a fitted random forest model. Our model suggested that the two most important aspects were length of the hint (with 80–160 words being ideal) and the reading level (with US grade level nine or lower, i.e. understandable by 14 year-olds or younger being ideal). Pedagogical aspects of the hints, based on feedback literacy theory [16, 64] were less important in the model; inclusion of alternate approaches to the solution were found to *decrease* a hint's rating, while including guidance beyond stating the answer *increased* a hint's rating – but the last item was the least influential of the four. The model showed no effect of sentiment on the hints' rating (most hints were positive, but in a wide range from slightly to very positive), and whether the hint highlighted a general rule (e.g. why the == operator cannot be used for string comparison in Java) also had no effect.

These results can provide useful lessons for educators and tool-makers about the best kind of hints to provide in contexts where short written hints are appropriate.

6.2 Hint generation

We asked educators to compare AI-generated hints from different AI models and different AI prompts, as well as human hints, without knowing which hints were generated by which method. We found that the best model in our study, GPT-4, produced hints that were rated more highly than hints produced by the [human!] researchers. This is promising for future research into adding hints in novice programming environments.

We found that there was as large a variation among prompts as there was among AI models. This is important for the automatic generation of hints, but also has implications for students' individual use of LLMs for help. Previous research [99] has found that non-experts can struggle to design prompts, so students may struggle to create a prompt themselves that produces a hint as good as the best hints in this study. This suggests that there is room for tools to "package up" pre-written

prompts and automatically deliver hints using these, rather than exposing the raw LLM prompt interface to users.

AI is currently undergoing rapid development, with new models being introduced every few months. In that regard, the specific models will outdate, and some of our results along with it. To ensure ours is a lasting contribution, we have detailed a reproducible method that allows a replication of the study to be run in future. Specifically: we described our process of hint creation, we detailed a methodology of running multiple prompts with multiple models, and how to use comparative judgement to evaluate these hints. All of our analysis scripts are in our OSF repository/supplementary materials (see subsection 4.1) to allow for easy replication.

Neither the researchers nor the AI models knew the exact context of the Snapshot (since this is not available in Blackbox), i.e. what precisely the student was aiming to do. All of them inferred the student's task based solely on the student's source code. This is in contrast to large portions of the hint-generating literature which rely on knowing the problem context to provide hints [15, 42, 63].

6.3 The missing student perspective

This study has only looked at educators' ranking of hints. Naturally, it would make sense to also investigate the student perspective of hints since they would be the ultimate consumers. We did not feel that students were likely to be able to rate hints in the same way given the complexity of the task: participants first need to read someone else's poorly-written code, understand what the code does and what the code is trying to do, understand the notes on the current problems with the code, and then read two hints and compare them to decide which would be the most useful. Our educator participants seemed able to complete this task (backed by their decades of experience). We were, however, not convinced that students could do the same, so we did not ask students to also perform this task.

A better design for evaluating hint quality for students might be to ask them to complete a given programming task, and when they get stuck, to be able to ask for a hint, like in the study by Roest et al. [86]. Students could be shown two hints and be asked to select the preferred one. This would remove the complexity of understanding someone else's code. Students would already know what they are trying to do, making evaluation of the usefulness of the hint more straightforward. It is possible to build the best prompt and model from this study into a tool to conduct such a second study.

With our current study design, it remains a possibility that our experienced educators are not very good at the task of deciding which hints would be most useful to students. For example, it may be that students prefer even shorter hints, or perhaps they favour more explanation. Perhaps students prefer being told the answer to receiving a hint. This is a classic educational conflict: who is best-placed to decide which hint is better? A student who perhaps wants the easy option of being told the answer or given a detailed explanation, or the educator who believes they are best-served by receiving a more circumspect hint? It is not obvious that either the student or the educator alone can provide the perfect assessment of which hint is best. In this study we have provided a large piece of the puzzle by asking educators.

6.4 The nature of hints

In this study we have chosen to focus on "one-shot" next-step hints which are provided to the student by a programming environment to help them move forwards. We have not considered any aspects of interface design, for example whether students should be able to manually request these hints at any time or whether they should be automatically offered or at what point. We consider these aspects to be outside the scope of this work, but they may be investigated in separate research.

Jeuring et al. [37] and Lohr et al. [56], for example, investigated when to provide hints, but found low agreement among educators.

We have also not considered the possibility of an ongoing dialogue between the student and the hint mechanism. One of the distinctive features of systems such as ChatGPT is the ability to converse with the LLM, asking for more detail, for clarification, or working in tandem together. Although part of our analysis was taken from feedback literacy theory [64], there are entire dimensions we omitted which are important for human contact but non-applicable in this kind of one-shot hint generation work, such as agency, direction, and temporality. All of these might be relevant in an ongoing dialogue. This is a potential avenue for future research.

6.5 The personal connection

The idea of an ongoing dialogue leads us back to the personal touch. We have focused on a very specific context: one-off next-step hint generation. Although we found that humans were not as good as the best AI on this task, this does not mean that human educators are redundant. The personal connection in education is still important. Portions of the hype around AI in education are reminisicent of the excitement around Massive Open Online Courses (MOOCs) a decade ago. Having the educational resources freely and widely available did not lead to a massive uptake or improvement in education, or to educators losing their jobs. There remains value in formal education based around human contact.

6.6 Relation to prior work

Our work provides interesting contrasts with some prior work. Compared to much previous work we found few issues with incorrect or misleading hints being generated by LLMs, which may reflect a difference in how we prompted the LLMs, or more likely general technological advancement in LLMs. Like prior work, we did find that LLMs did not always obey our instructions. Despite asking for only a hint, several "hints" provided exact solutions, and many would list out all the problems in the code despite explicitly being asked for a single hint relating to a one selected problem.

Our prompts were quite different to some previous work. For example, Roest et al. [86] used very minimal requests of 1–2 sentences alongside the problem description and code. Our prompts are much longer, but also have no problem description to work with, which is in contrast to the majority of previous work.

In their analysis of enhanced error explanations using feedback literacy theory, Cucuiat and Waite [16] found that explanations using *guiding* were preferred to *telling* (feedback literacy theory term [64]) which matched with our results. However they found that educators considered *developed understanding* as positive, but under our operationalisation (where this meant that the hint explained the general rule, e.g. why you cannot use the == operator for string comparison in Java) it made no difference to how the hints were evaluated. This is particularly interesting because Sheese et al. [90] found that students would not typically seek out the general rule for themselves, and educators seem to think that it is also not worthwhile to include it in a next-step hint. This is also in contrast to recommendations of Aleven et al. [3] who suggested that hints should state the domain principle.

Opening up a different perspective, the highest level of abstraction in feedback literacy theory as used by Cucuiat and Waite, was considered negative in our hints. This may suggest that educators, when considering concrete examples of next-step hints, consider this too overwhelming to be helpful overall.

6.7 Comparative judgement as a research method

There were multiple research methods which could have been used to get educators' opinions on the hints. One option would be to interview them, as done by Cucuiat and Waite [16]. This has the advantage of getting deeper answers about the why, but it would also not have allowed us to end up with precise guidance over the hint length or the reading level. The use of comparative judgement (plus a survey to fill in or corroborate the why) plus a random forest for analysis allowed us to exact specific guidelines on what made a good hint. Our checks verified that, at least for this length of task, participants took the task seriously, showed essentially no signs of boredom or giving up, and produced reliable results. No participant mentioned being confused by the requirements of the task. We believe comparative judgement may be a useful research method for the future for asking participants to rank a set of stimuli.

Another advantage of asking participants to rank hints is as follows. It is possible that there is a discrepancy of an educator's preference expressed in the abstract and their opinion when confronted with a concrete representation of the concept. For example, educators may express a general preference to *opening up perspectives* when interacting with students, but rate hints attempting to do just that lower, because the associated drawbacks (length, complexity, distraction) become more obvious. In this light, comparative judgement as used here can offer more concrete results, by observing what participants "do" (how they rank) rather than what they "say" (when asked in an interview).

7 Threats to Validity

In this section we detail various threats to validity and limitations of our study.

7.1 Low completion rate

Our completion rate was relatively low: only around 50% of those who signed up to participate in the ranking task went on to complete it. One possible explanation is that the task itself was too boring or hard for participants. However, the comparative judgement platform allows us to see who began the task, and only three participants began the task and did not finish, so the non-completers did not even start the task.

It is possible the task description was off-putting. However, it consisted of only 2-3 relatively sparse pages (available in our OSF repository/supplementary materials, see subsection 4.1). We believe the most likely explanation is the time of year we recruited and the general business of academics and teachers. We had aimed to recruit before teaching began but we ended up recruiting in August and September when many high school and university teachers in the northern hemisphere are very busy with the start of their teaching terms.

7.2 Low number of snapshots

Due to the low participation rate, we were only able to get enough educators to rate four code Snapshots (despite having eight available to rate). This means that our results may be affected by the low number of Snapshots that we had available to us. All of our material is freely available in our OSF repository/supplementary materials, and we would welcome replication/extension attempts to rate more Snapshots.

7.3 Snapshot selection and notes

The Snapshot selection was performed by one researcher only. There is a risk that an unrepresentative set of Snapshots was produced. Firstly, this would not necessarily invalidate the study: they are still real student issues on which to practice generating hints. Secondly, educators commented

unprompted on the selection being a good set of examples, so we have some validation that this was a good set of examples.

Another possible related issue is the "notes" section we provided beneath each example to the participants. We believed that if participants were shown the code without any notes, they may have had to spend a long time spotting the issues for themselves, and their ranking would be based on whether or not they had spotted all the issues themselves. To give a "level playing field" we decided to tell all the participants what the issues were based on our own expert assessment, at the risk of stopping educators engaging deeply with the code themselves. These notes could have influenced the hint ranking, perhaps by making educators downrate hints which did not address all the problems. However, we trusted the experience of our educators to decide whether they felt all issues should be addressed (several commented in the survey about whether hints should or should not address all issues, so it was something they were thinking about). Furthermore, it was not clear how to decide whether the notes were an influence without introducing another factor into our 5x4 design which would have made the experiment untenable.

Finally, another potential issue is the identification of incorrectness in our hints. This was a minor *post hoc* analysis. It was performed by one of the research team with over twenty years of professional programming experience, but it is possible that they missed something in the other hints. All of them are available in our public repository for further checking.

7.4 Comparative judgement

The comparative judgement technique we used assumes that there is a single underlying ranking of the hints which all judges (here: educators) can agree on. If this is not the case then the results (which rely on the hint ranking) could be impacted or meaningless. However, the "infit" consistency metrics that is used to check this in comparative judgement showed that in general consistency was good (see Figure 4).

7.5 Random forest model performance

The random forest model that we have used to investigate the importance of hint characteristics had a cross-validated R^2 of 0.42. There are no general guidelines for interpreting R^2 so we cannot provide a simple categorisation as to whether this value is good, fair or poor. Nevertheless, it is not perfect, and there may be other hint characteristics which we did not model which would improve the performance of our modelling. These might be features that are hard to categorise such as clarity of prose, or it might be alternate measures of our existing characteristics (e.g. an alternative readability metric, or character length rather than word length). Our data on the ranks, as well as the hints themselves are fully public if other researchers would like to investigate constructing a superior model.

7.6 Readability versus target demographic

Our instructions suggested that educators should imagine a 16–18 year old demographic, i.e. US school grade 10–12. We find that hint readability drops off after grade 10. So it may be that our result was determined by our study instructions. It is difficult to find national readability statistics in terms of grade levels (large-scale surveys like PIAAC use different metrics [46]). American guidelines for writing health information suggest that below grade 6 is easy, 7–9 is average difficulty, and 10 or above is difficult [21]. This would match well with our results in Figure 11 that show a drop at 7.5–10 followed by a larger drop-off. Had we chosen a younger demographic it is likely that our results would have an earlier drop-off – but it may be unlikely that an older demographic would have encouraged more complex hints.

7.7 Outdated LLMs

The LLMs we used will naturally outdate as technology progresses, but we have tried to mitigate this through our research design that used multiple models and multiple prompts. Our findings can also inform the hints literature independently of the technology used to generate the hints for the study.

7.8 Reproducible prompt formation process

One of our aims was to detail a reproducible method for conducting this study, in order that it could be repeated in future with newer LLMs. In particular, this may require new prompts to be created that are effective with these future LLMs. We used an approach of independent creation of these prompts in line with the brainstorming literature but it turned out to be a more *ad hoc* process than anticipated. We are not aware of guidelines in the literature on how to systematically create a prompt in a research context, and many existing papers that need a prompt simply state one without providing any details on its creation. At the present time it is more of a creative process (akin to writing prose) rather than a systematic process (more akin to writing straightforward program code). We encourage researchers interested in LLMs to consider how to make the prompt formation process more systematic, or at least provide guidelines on doing so.

8 Future work

One clear future direction is to ask students to evaluate hints. We believe the best study design will be to implement automatic hint generation in a novice programming environment and then ask students to use it and rate or rank the hints they are given, and/or monitor their programming activity immediately after receiving the hint.

Although we believe we have shown that comparative judgement is a viable experimental technique, the fact remains that recruiting participants (especially busy teachers and academics) is a difficult process; we had eight Snapshots prepared but only recruited enough completing participants to evaluate four of them. Some recent work [33, 55] has investigated whether LLMs can emulate human participants in social science experiments in order to generate synthetic data. On the one hand, this risks AI "marking its own work", with LLMs evaluating the output of LLMs (as done, for example, by Koutcheme et al. [49, 50]). On the other hand there may be ways to use this technique to boost evaluation of new behavioural interventions such as identifying better hints.

The hint generation in this work was done with a set of brainstormed prompts. We not only know which prompt produced the best hints, we also now have extra information about the characteristics of best hints, in terms of length, reading level and other aspects (e.g. that offering alternative approaches is rated negatively). This opens the possibility to design a new prompt that takes these insights into account in order to improve hint generation. This minor step forward is one possible avenue to deploy LLM evaluation, rather than recruiting 85 human participants again just to test a minor improvement on the prompts.

One further direction for related work is to analyse our existing data under other hint classifications. Although we chose to focus on feedback literacy theory, other classifications have been proposed, such as by Keuning et al. [42] and separately by Suzuki et al. [93]. Categorising the existing hints using those schemes and relating the results to our ranking is a possible next step. With our data being open, this can also be done by other research teams.

9 Conclusion

In this paper we asked human educators to rank sets of hints generated by AI models and human researchers using comparative judgement. This provided findings in several different dimensions.

One finding is that GPT-4 was found to produce better hints than experienced humans when using our best prompt. It is particularly important to note that the hints were generated without providing any context of the task that the student was performing. The data was taken from the Blackbox dataset, which provides examples from arbitrary novice programmers, without knowledge of the exact task being performed. In this GPT-4-better-than-humans sense, this paper is one in a recent line of "LLMs beat humans at < programming education task >". Prior results in this line examined creating code explanations [52], small programming exercises [44] or full programming exams [58]. However, we provide several further contributions beyond this latest LLM feat.

Some do relate specifically to the operation of LLMs. We evaluated five different LLM prompts which are quite different in their construction, including two multi-stage prompts. Although the prompts do have an interaction with the choice of LLM, one was clearly better than the others (prompt 3 in Table 1). This prompt first asked the LLM to summarise the task the student was inferred to perform and then fed the result back to a second request to provide a hint. (Our prompts, methods and analysis scripts are all open to allow easy replication in future as LLMs advance.) Furthermore, although GPT-4 was better than humans, all the other models were not. There is a pronounced effect of model, prompt and their interaction, which showed much greater variation in average performance than we found between the five human researchers who created hints. It is still the case that LLMs beat humans only with the right model and the right prompt.

We also provide contributions that are entirely orthogonal to AI. Our study can be seen solely as an investigation into the characteristics of hints that are most important to judge the hint useful – with the fact that some hints were generated by AI merely acting as a convenient artifact generation mechanism. We have found that the most important attributes predicting a hint's ranking was its length and reading level. Experienced Java educators (more than half with an equivalent of over 20+ years of experience) rated hints most highly where the word count was 80–160 words, the reading level was typically understandable by those in US grade 9 (age 14) or below, where guidance was provided beyond just stating the answer, but alternative approaches to solving the problem were avoided. We found no effect of sentiment or of explaining a more general underlying principle on the perceived quality of a hint.

We have also demonstrated the use of comparative judgement (previously primarily used for assessing writing skills) as a research methodology, showing that, at least for a 20 minute task, participants took it seriously and did not get bored, and the results produced were reliable and interpretable. Comparative judgement is useful when participants are required, individually or collectively, to rank a number of experimental stimuli than can be placed alongside each other on one screen. There are several free comparative judgement websites; we used NoMoreMarking[4], with details on how we set up the task available in our OSF repository/supplementary materials (along with all of our data and analysis code) as described in subsection 4.1.

Acknowledgments

We are very grateful to all of our participants for taking part, despite being busy educators. We thank Arto Hellas for his comments on the study design, Jane Waite for offering guidance on feedback literacy, Barbara Ericson for her help with study recruitment, and the NoMoreMarking team for providing the Comparative Judgement platform. Juho Leinonen was supported in this research by the Research Council of Finland (Academy Research Fellow grant number 356114). Finally, we are incredibly grateful to our reviewers, especially the fabled reviewer #2 who wrote over 4000 words on how to improve the paper. We hope we did them justice.

[4] https://www.nomoremarking.com/

References

[1] Alireza Ahadi, Raymond Lister, Heikki Haapala, and Arto Vihavainen. 2015. Exploring Machine Learning Methods to Automatically Identify Students in Need of Assistance. In *Proceedings of the Eleventh Annual International Conference on International Computing Education Research* (Omaha, Nebraska, USA) *(ICER '15)*. Association for Computing Machinery, New York, NY, USA, 121–130. doi:10.1145/2787622.2787717

[2] Umair Z. Ahmed, Nisheeth Srivastava, Renuka Sindhgatta, and Amey Karkare. 2020. Characterizing the Pedagogical Benefits of Adaptive Feedback for Compilation Errors by Novice Programmers. In *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering: Software Engineering Education and Training* (Seoul, South Korea) *(ICSE-SEET '20)*. Association for Computing Machinery, New York, NY, USA, 139–150. doi:10.1145/3377814.3381703

[3] Vincent Aleven, Ido Roll, Bruce M. McLaren, and Kenneth R. Koedinger. 2016. Help Helps, But Only So Much: Research on Help Seeking with Intelligent Tutoring Systems. *International Journal of Artificial Intelligence in Education* 26, 1 (01 Mar 2016), 205–223. doi:10.1007/s40593-015-0089-1

[4] André Altmann, Laura Toloşi, Oliver Sander, and Thomas Lengauer. 2010. Permutation importance: a corrected feature importance measure. *Bioinformatics* 26, 10 (2010), 1340–1347.

[5] S Bartholomew and Emily Yoshikawa-Ruesch. 2018. A systematic review of research around adaptive comparative judgement (ACJ) in K-16 education. *Council on Technology an Engineering Teacher Education: Research Monograph Series* 1, 1 (2018). doi:10.21061/ctete-rms.v1.c.1

[6] Anastasiia Birillo, Elizaveta Artser, Anna Potriasaeva, Ilya Vlasov, Katsiaryna Dzialets, Yaroslav Golubev, Igor Gerasimov, Hieke Keuning, and Timofey Bryksin. 2024. One Step at a Time: Combining LLMs and Static Analysis to Generate Next-Step Hints for Programming Tasks. In *Proceedings of the 24th Koli Calling International Conference on Computing Education Research (Koli Calling '24)*. Association for Computing Machinery, New York, NY, USA, Article 9, 12 pages. doi:10.1145/3699538.3699556

[7] Brad Boehmke and Brandon M Greenwell. 2019. *Hands-on machine learning with R*. Chapman and Hall/CRC.

[8] Neil C. C. Brown and Amjad Altadmri. 2017. Novice Java Programming Mistakes: Large-Scale Data vs. Educator Beliefs. *ACM Trans. Comput. Educ.* 17, 2, Article 7 (May 2017), 21 pages. doi:10.1145/2994154

[9] Neil C. C. Brown, Jamie Ford, Pierre Weill-Tessier, and Michael Kölling. 2023. Quick Fixes for Novice Programmers: Effective but Under-Utilised. In *Proceedings of the 2023 Conference on United Kingdom & Ireland Computing Education Research (UKICER '23)*. Association for Computing Machinery, New York, NY, USA, Article 3, 7 pages. doi:10.1145/3610969.3611117

[10] Neil C. C. Brown, Michael Kölling, Davin McCall, and Ian Utting. 2014. Blackbox: A Large Scale Repository of Novice Programmers' Activity. In *Proceedings of the 45th ACM Technical Symposium on Computer Science Education* (Atlanta, Georgia, USA) *(SIGCSE '14)*. Association for Computing Machinery, New York, NY, USA, 223–228. doi:10.1145/2538862.2538924

[11] Marc Brysbaert. 2019. How many words do we read per minute? A review and meta-analysis of reading rate. *Journal of Memory and Language* 109 (2019), 104047. doi:10.1016/j.jml.2019.104047

[12] Francisco Enrique Vicente Castro and Kathi Fisler. 2020. Qualitative Analyses of Movements Between Task-Level and Code-Level Thinking of Novice Programmers. In *Proceedings of the 51st ACM Technical Symposium on Computer Science Education* (Portland, OR, USA) *(SIGCSE '20)*. Association for Computing Machinery, New York, NY, USA, 487–493. doi:10.1145/3328778.3366847

[13] Victoria Clarke and Virginia Braun. 2017. Thematic analysis. *The journal of positive psychology* 12, 3 (2017), 297–298.

[14] Albert T Corbett and John R Anderson. 2001. Locus of feedback control in computer-based tutoring: Impact on learning rate, achievement and attitudes. In *Proceedings of the SIGCHI conference on Human factors in computing systems*. 245–252.

[15] Tyne Crow, Andrew Luxton-Reilly, and Burkhard Wuensche. 2018. Intelligent Tutoring Systems for Programming Education: A Systematic Review. In *Proceedings of the 20th Australasian Computing Education Conference* (Brisbane, Queensland, Australia) *(ACE '18)*. Association for Computing Machinery, New York, NY, USA, 53–62. doi:10.1145/3160489.3160492

[16] Veronica Cucuiat and Jane Waite. 2024. Feedback Literacy: Holistic Analysis of Secondary Educators' Views of LLM Explanations of Program Error Messages. In *Proceedings of the 2024 on Innovation and Technology in Computer Science Education V. 1* (Milan, Italy) *(ITiCSE 2024)*. Association for Computing Machinery, New York, NY, USA, 192–198. doi:10.1145/3649217.3653595

[17] Paul Denny, Juho Leinonen, James Prather, Andrew Luxton-Reilly, Thezyrie Amarouche, Brett A. Becker, and Brent N. Reeves. 2024. Prompt Problems: A New Programming Exercise for the Generative AI Era. In *Proceedings of the 55th ACM Technical Symposium on Computer Science Education V. 1* (Portland, OR, USA) *(SIGCSE 2024)*. Association for Computing Machinery, New York, NY, USA, 296–302. doi:10.1145/3626252.3630909

[18] Paul Denny, Stephen MacNeil, Jaromir Savelka, Leo Porter, and Andrew Luxton-Reilly. 2024. Desirable Characteristics for AI Teaching Assistants in Programming Education. In *Proceedings of the 2024 on Innovation and Technology in*

Computer Science Education V. 1 (Milan, Italy) *(ITiCSE 2024)*. Association for Computing Machinery, New York, NY, USA, 408–414. doi:10.1145/3649217.3653574

[19] Paul Denny, James Prather, Brett A. Becker, James Finnie-Ansley, Arto Hellas, Juho Leinonen, Andrew Luxton-Reilly, Brent N. Reeves, Eddie Antonio Santos, and Sami Sarsa. 2024. Computing Education in the Era of Generative AI. *Commun. ACM* 67, 2 (Jan. 2024), 56–67. doi:10.1145/3624720

[20] Paul Denny, James Prather, Brett A. Becker, Catherine Mooney, John Homer, Zachary C Albrecht, and Garrett B. Powell. 2021. On Designing Programming Error Messages for Novices: Readability and Its Constituent Factors. In *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems* (Yokohama, Japan) *(CHI '21)*. Association for Computing Machinery, New York, NY, USA, Article 55, 15 pages. doi:10.1145/3411764.3445696

[21] Matthew R. Edmunds, Robert J. Barry, and Alastair K. Denniston. 2013. Readability Assessment of Online Ophthalmic Patient Information. *JAMA Ophthalmology* 131, 12 (12 2013), 1610–1616. doi:10.1001/jamaophthalmol.2013.5521 arXiv:https://jamanetwork.com/journals/jamaophthalmology/articlepdf/1763220/eoi130179.pdf

[22] Nickolas J.G. Falkner and Katrina E. Falkner. 2012. A Fast Measure for Identifying At-Risk Students in Computer Science. In *Proceedings of the Ninth Annual International Conference on International Computing Education Research* (Auckland, New Zealand) *(ICER '12)*. Association for Computing Machinery, New York, NY, USA, 55–62. doi:10.1145/2361276.2361288

[23] Alexander J. Fiannaca, Chinmay Kulkarni, Carrie J Cai, and Michael Terry. 2023. Programming without a Programming Language: Challenges and Opportunities for Designing Developer Tools for Prompt Programming. In *Extended Abstracts of the 2023 CHI Conference on Human Factors in Computing Systems* (Hamburg, Germany) *(CHI EA '23)*. Association for Computing Machinery, New York, NY, USA, Article 235, 7 pages. doi:10.1145/3544549.3585737

[24] Davide Fossati, Barbara Di Eugenio, STELLAN Ohlsson, Christopher Brown, and Lin Chen. 2015. Data driven automatic feedback generation in the iList intelligent tutoring system. *Technology, Instruction, Cognition and Learning* 10, 1 (2015), 5–26.

[25] Sandy Garner, Patricia Haden, and Anthony Robins. 2005. My Program is Correct but It Doesn't Run: A Preliminary Investigation of Novice Programmers' Problems. In *Proceedings of the 7th Australasian Conference on Computing Education - Volume 42* (Newcastle, New South Wales, Australia) *(ACE '05)*. Australian Computer Society, Inc., AUS, 173–180.

[26] Aashish Ghimire and John Edwards. 2024. Coding with AI: How Are Tools Like ChatGPT Being Used by Students in Foundational Programming Courses. In *Artificial Intelligence in Education*, Andrew M. Olney, Irene-Angelica Chounta, Zitao Liu, Olga C. Santos, and Ig Ibert Bittencourt (Eds.). Springer Nature Switzerland, Cham, 259–267.

[27] Elena L. Glassman, Aaron Lin, Carrie J. Cai, and Robert C. Miller. 2016. Learnersourcing Personalized Hints. In *Proceedings of the 19th ACM Conference on Computer-Supported Cooperative Work & Social Computing* (San Francisco, California, USA) *(CSCW '16)*. Association for Computing Machinery, New York, NY, USA, 1626–1636. doi:10.1145/2818048.2820011

[28] Philip J. Guo, Julia M. Markel, and Xiong Zhang. 2020. Learnersourcing at Scale to Overcome Expert Blind Spots for Introductory Programming: A Three-Year Deployment Study on the Python Tutor Website. In *Proceedings of the Seventh ACM Conference on Learning @ Scale* (Virtual Event, USA) *(L@S '20)*. Association for Computing Machinery, New York, NY, USA, 301–304. doi:10.1145/3386527.3406733

[29] Luke Gusukuma, Dennis Kafura, and Austin Cory Bart. 2017. Authoring feedback for novice programmers in a block-based language. In *2017 IEEE Blocks and Beyond Workshop (B&B)*. 37–40. doi:10.1109/BLOCKS.2017.8120407

[30] Arto Hellas, Petri Ihantola, Andrew Petersen, Vangel V. Ajanovski, Mirela Gutica, Timo Hynninen, Antti Knutas, Juho Leinonen, Chris Messom, and Soohyun Nam Liao. 2018. Taxonomizing Features and Methods for Identifying At-Risk Students in Computing Courses. In *Proceedings of the 23rd Annual ACM Conference on Innovation and Technology in Computer Science Education* (Larnaca, Cyprus) *(ITiCSE 2018)*. Association for Computing Machinery, New York, NY, USA, 364–365. doi:10.1145/3197091.3205845

[31] Arto Hellas, Juho Leinonen, Sami Sarsa, Charles Koutcheme, Lilja Kujanpää, and Juha Sorva. 2023. Exploring the Responses of Large Language Models to Beginner Programmers' Help Requests. In *Proceedings of the 2023 ACM Conference on International Computing Education Research - Volume 1* (Chicago, IL, USA) *(ICER '23)*. Association for Computing Machinery, New York, NY, USA, 93–105. doi:10.1145/3568813.3600139

[32] Katharina Hellman and Matthias Nuckles. 2013. Expert blind spot in pre-service and in-service mathematics teachers: task design moderates overestimation of novices' performance. In *Proceedings of the Annual Meeting of the Cognitive Science Society*, Vol. 35.

[33] Luke Hewitt, Ashwini Ashokkumar, Isaias Ghezae, and Robb Willer. 2024. *Predicting Results of Social Science Experiments Using Large Language Models*. Technical Report. Working Paper. https://samim.io/dl/Predicting%20results%20of%20social%20science%20experiments%20using%20large%20language%20models.pdf

[34] Tin Kam Ho. 1995. Random decision forests. In *Proceedings of the Third International Conference on Document Analysis and Recognition (Volume 1) - Volume 1 (ICDAR '95)*. IEEE Computer Society, USA, 278.

[35] C. Hutto and Eric Gilbert. 2014. VADER: A Parsimonious Rule-Based Model for Sentiment Analysis of Social Media Text. *Proceedings of the International AAAI Conference on Web and Social Media* 8, 1 (May 2014), 216–225. doi:10.1609/icwsm.v8i1.14550

[36] Michelle Ichinco and Caitlin Kelleher. 2018. Semi-Automatic Suggestion Generation for Young Novice Programmers in an Open-Ended Context. In *Proceedings of the 17th ACM Conference on Interaction Design and Children* (Trondheim, Norway) *(IDC '18)*. Association for Computing Machinery, New York, NY, USA, 405–412. doi:10.1145/3202185.3202762

[37] Johan Jeuring, Hieke Keuning, Samiha Marwan, Dennis Bouvier, Cruz Izu, Natalie Kiesler, Teemu Lehtinen, Dominic Lohr, Andrew Peterson, and Sami Sarsa. 2022. Towards Giving Timely Formative Feedback and Hints to Novice Programmers. In *Proceedings of the 2022 Working Group Reports on Innovation and Technology in Computer Science Education* (Dublin, Ireland) *(ITiCSE-WGR '22)*. Association for Computing Machinery, New York, NY, USA, 95–115. doi:10.1145/3571785.3574124

[38] Ian Jones and Ben Davies. 2024. Comparative judgement in education research. *International Journal of Research & Method in Education* 47, 2 (2024), 170–181. doi:10.1080/1743727X.2023.2242273 arXiv:https://doi.org/10.1080/1743727X.2023.2242273

[39] Ishika Joshi, Ritvik Budhiraja, Pranav Deepak Tanna, Lovenya Jain, Mihika Deshpande, Arjun Srivastava, Srinivas Rallapalli, Harshal D Akolekar, Jagat Sesh Challa, and Dhruv Kumar. 2023. "With Great Power Comes Great Responsibility!": Student and Instructor Perspectives on the influence of LLMs on Undergraduate Engineering Education. arXiv:2309.10694 [cs.HC]

[40] Majeed Kazemitabaar, Justin Chow, Carl Ka To Ma, Barbara J. Ericson, David Weintrop, and Tovi Grossman. 2023. Studying the Effect of AI Code Generators on Supporting Novice Learners in Introductory Programming. In *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems* (Hamburg, Germany) *(CHI '23)*. Association for Computing Machinery, New York, NY, USA, Article 455, 23 pages. doi:10.1145/3544548.3580919

[41] Tyson Kendon, Leanne Wu, and John Aycock. 2023. AI-Generated Code Not Considered Harmful. In *Proceedings of the 25th Western Canadian Conference on Computing Education* (Vancouver, BC, Canada) *(WCCCE '23)*. Association for Computing Machinery, New York, NY, USA, Article 3, 7 pages. doi:10.1145/3593342.3593349

[42] Hieke Keuning, Johan Jeuring, and Bastiaan Heeren. 2018. A Systematic Literature Review of Automated Feedback Generation for Programming Exercises. *ACM Trans. Comput. Educ.* 19, 1, Article 3 (Sept. 2018), 43 pages. doi:10.1145/3231711

[43] Natalie Kiesler, Dominic Lohr, and Hieke Keuning. 2023. Exploring the Potential of Large Language Models to Generate Formative Programming Feedback. In *2023 IEEE Frontiers in Education Conference (FIE)*. 1–5. doi:10.1109/FIE58773.2023.10343457

[44] Natalie Kiesler and Daniel Schiffner. 2023. Large Language Models in Introductory Programming Education: ChatGPT's Performance and Implications for Assessments. arXiv:2308.08572 [cs.SE] https://arxiv.org/abs/2308.08572

[45] JP Kincaid. 1975. Derivation of new readability formulas (automated readability index, fog count and flesch reading ease formula) for navy enlisted personnel. *Chief of Naval Technical Training* (1975).

[46] Irwin Kirsch and Mary Louise Lennon. 2017. PIAAC: A new design for a new era. *Large-scale assessments in education* 5 (2017), 1–22.

[47] Kenneth R Koedinger, Albert T Corbett, and Charles Perfetti. 2012. The Knowledge-Learning-Instruction framework: Bridging the science-practice chasm to enhance robust student learning. *Cognitive science* 36, 5 (2012), 757–798.

[48] MJ Kolen and RL Brennan. 2016. 'No More Marking': An online tool for comparative judgement. *ISSN 1756-509X* (2016), 12.

[49] Charles Koutcheme, Nicola Dainese, Sami Sarsa, Arto Hellas, Juho Leinonen, Syed Ashraf, and Paul Denny. 2025. Evaluating Language Models for Generating and Judging Programming Feedback. In *Proceedings of the 56th ACM Technical Symposium on Computer Science Education V. 1* (Pittsburgh, PA, USA) *(SIGCSETS 2025)*. Association for Computing Machinery, New York, NY, USA, 624–630. doi:10.1145/3641554.3701791

[50] Charles Koutcheme, Nicola Dainese, Sami Sarsa, Arto Hellas, Juho Leinonen, and Paul Denny. 2024. Open Source Language Models Can Provide Feedback: Evaluating LLMs' Ability to Help Students Using GPT-4-As-A-Judge. In *Proceedings of the 2024 on Innovation and Technology in Computer Science Education V. 1* (Milan, Italy) *(ITiCSE 2024)*. Association for Computing Machinery, New York, NY, USA, 52–58. doi:10.1145/3649217.3653612

[51] Sam Lau and Philip Guo. 2023. From "Ban It Till We Understand It" to "Resistance is Futile": How University Programming Instructors Plan to Adapt as More Students Use AI Code Generation and Explanation Tools Such as ChatGPT and GitHub Copilot. In *Proceedings of the 2023 ACM Conference on International Computing Education Research - Volume 1* (Chicago, IL, USA) *(ICER '23)*. Association for Computing Machinery, New York, NY, USA, 106–121. doi:10.1145/3568813.3600138

[52] Juho Leinonen, Paul Denny, Stephen MacNeil, Sami Sarsa, Seth Bernstein, Joanne Kim, Andrew Tran, and Arto Hellas. 2023. Comparing Code Explanations Created by Students and Large Language Models. In *Proceedings of the 2023 Conference on Innovation and Technology in Computer Science Education V. 1* (Turku, Finland) *(ITiCSE 2023)*. Association

for Computing Machinery, New York, NY, USA, 124–130. doi:10.1145/3587102.3588785

[53] Juho Leinonen, Arto Hellas, Sami Sarsa, Brent Reeves, Paul Denny, James Prather, and Brett A. Becker. 2023. Using Large Language Models to Enhance Programming Error Messages. In *Proceedings of the 54th ACM Technical Symposium on Computer Science Education V. 1* (Toronto ON, Canada) *(SIGCSE 2023)*. Association for Computing Machinery, New York, NY, USA, 563–569. doi:10.1145/3545945.3569770

[54] Mark Liffiton, Brad E Sheese, Jaromir Savelka, and Paul Denny. 2024. CodeHelp: Using Large Language Models with Guardrails for Scalable Support in Programming Classes. 11 pages. doi:10.1145/3631802.3631830

[55] Steffen Lippert, Anna Dreber, Magnus Johannesson, Warren Tierney, Wilson Cyrus-Lai, Eric Luis Uhlmann, Emotion Expression Collaboration, and Thomas Pfeiffer. 2024. Can large language models help predict results from a complex behavioural science study? *Royal Society Open Science* 11, 9 (2024), 240682.

[56] Dominic Lohr, Natalie Kiesler, Hieke Keuning, and Johan Jeuring. 2024. "Let Them Try to Figure It Out First" - Reasons Why Experts (Do Not) Provide Feedback to Novice Programmers. In *Proceedings of the 2024 on Innovation and Technology in Computer Science Education V. 1* (Milan, Italy) *(ITiCSE 2024)*. Association for Computing Machinery, New York, NY, USA, 38–44. doi:10.1145/3649217.3653530

[57] Stephen MacNeil, Andrew Tran, Arto Hellas, Joanne Kim, Sami Sarsa, Paul Denny, Seth Bernstein, and Juho Leinonen. 2023. Experiences from Using Code Explanations Generated by Large Language Models in a Web Software Development E-Book. In *Proceedings of the 54th ACM Technical Symposium on Computer Science Education V. 1* (Toronto ON, Canada) *(SIGCSE 2023)*. Association for Computing Machinery, New York, NY, USA, 931–937. doi:10.1145/3545945.3569785

[58] Joyce Mahon, Brian Mac Namee, and Brett A. Becker. 2023. No More Pencils No More Books: Capabilities of Generative AI on Irish and UK Computer Science School Leaving Examinations. In *Proceedings of the 2023 Conference on United Kingdom & Ireland Computing Education Research* (Swansea, Wales Uk) *(UKICER '23)*. Association for Computing Machinery, New York, NY, USA, Article 2, 7 pages. doi:10.1145/3610969.3610982

[59] Alina Mailach, Dominik Gorgosch, Norbert Siegmund, and Janet Siegmund. 2024. "Ok Pal, we have to code that now": interaction patterns of programming beginners with a conversational chatbot. *Empirical Software Engineering* 30, 1 (30 Nov 2024), 34. doi:10.1007/s10664-024-10561-6

[60] Samiha Marwan, Joseph Jay Williams, and Thomas Price. 2019. An Evaluation of the Impact of Automated Programming Hints on Performance and Learning. In *Proceedings of the 2019 ACM Conference on International Computing Education Research* (Toronto ON, Canada) *(ICER '19)*. Association for Computing Machinery, New York, NY, USA, 61–70. doi:10.1145/3291279.3339420

[61] Samiha Marwan, Nicholas Lytle, Joseph Jay Williams, and Thomas Price. 2019. The Impact of Adding Textual Explanations to Next-Step Hints in a Novice Programming Environment. In *Proceedings of the 2019 ACM Conference on Innovation and Technology in Computer Science Education* (Aberdeen, Scotland Uk) *(ITiCSE '19)*. Association for Computing Machinery, New York, NY, USA, 520–526. doi:10.1145/3304221.3319759

[62] Samiha Marwan and Thomas W. Price. 2023. ISnap: Evolution and Evaluation of a Data-Driven Hint System for Block-Based Programming. *IEEE Trans. Learn. Technol.* 16, 3.2 (June 2023), 399–413. doi:10.1109/TLT.2022.3223577

[63] Jessica McBroom, Irena Koprinska, and Kalina Yacef. 2021. A Survey of Automated Programming Hint Generation: The HINTS Framework. *ACM Comput. Surv.* 54, 8, Article 172 (oct 2021), 27 pages. doi:10.1145/3469885

[64] Angela J McLean, Carol H Bond, and Helen D Nicholson. 2015. An anatomy of feedback: a phenomenographic investigation of undergraduate students' conceptions of feedback. *Studies in Higher Education* 40, 5 (2015), 921–932.

[65] Daye Nam, Andrew Macvean, Vincent Hellendoorn, Bogdan Vasilescu, and Brad Myers. 2024. Using an LLM to Help With Code Understanding. In *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering* (Lisbon, Portugal) *(ICSE '24)*. Association for Computing Machinery, New York, NY, USA, Article 97, 13 pages. doi:10.1145/3597503.3639187

[66] Susanne Narciss. 2008. Feedback strategies for interactive learning tasks. In *Handbook of research on educational communications and technology*. Routledge, 125–143.

[67] Mitchell J Nathan, Kenneth R Koedinger, Martha W Alibali, et al. 2001. Expert blind spot: When content knowledge eclipses pedagogical content knowledge. In *Proceedings of the third international conference on cognitive science*, Vol. 644648. 644–648.

[68] Ha Nguyen and Vicki Allan. 2024. Using GPT-4 to Provide Tiered, Formative Code Feedback. In *Proceedings of the 55th ACM Technical Symposium on Computer Science Education V. 1* (Portland, OR, USA) *(SIGCSE 2024)*. Association for Computing Machinery, New York, NY, USA, 958–964. doi:10.1145/3626252.3630960

[69] Sydney Nguyen, Hannah McLean Babe, Yangtian Zi, Arjun Guha, Carolyn Jane Anderson, and Molly Q Feldman. 2024. How Beginning Programmers and Code LLMs (Mis)read Each Other. In *Proceedings of the 2024 CHI Conference on Human Factors in Computing Systems* (Honolulu, HI, USA) *(CHI '24)*. Association for Computing Machinery, New York, NY, USA, Article 651, 26 pages. doi:10.1145/3613904.3642706

[70] Florian Obermüller, Ute Heuer, and Gordon Fraser. 2021. Guiding Next-Step Hint Generation Using Automated Tests. In *Proceedings of the 26th ACM Conference on Innovation and Technology in Computer Science Education V. 1* (Virtual

Event, Germany) *(ITiCSE '21)*. Association for Computing Machinery, New York, NY, USA, 220–226. doi:10.1145/3430665.3456344

[71] Benjamin Paaßen, Jessica McBroom, Bryn Jeffries, Irena Koprinska, and Kalina Yacef. 2021. Next Steps for Next-step Hints: Lessons Learned from Teacher Evaluations of Automatic Programming Hints. In *Joint Proceedings of the Workshops of the 14th International Conference on Educational Data Mining (EDM)*.

[72] Maciej Pankiewicz and Ryan S. Baker. 2024. Navigating Compiler Errors with AI Assistance - A Study of GPT Hints in an Introductory Programming Course. In *Proceedings of the 2024 on Innovation and Technology in Computer Science Education V. 1* (Milan, Italy) *(ITiCSE 2024)*. Association for Computing Machinery, New York, NY, USA, 94–100. doi:10.1145/3649217.3653608

[73] Phitchaya Mangpo Phothilimthana and Sumukh Sridhara. 2017. High-Coverage Hint Generation for Massive Courses: Do Automated Hints Help CS1 Students?. In *Proceedings of the 2017 ACM Conference on Innovation and Technology in Computer Science Education* (Bologna, Italy) *(ITiCSE '17)*. Association for Computing Machinery, New York, NY, USA, 182–187. doi:10.1145/3059009.3059058

[74] Alastair Pollitt. 2012. The method of Adaptive Comparative Judgement. *Assessment in Education: Principles, Policy & Practice* 19, 3 (2012), 281–300. doi:10.1080/0969594X.2012.665354 arXiv:https://doi.org/10.1080/0969594X.2012.665354

[75] Stanislav Pozdniakov, Jonathan Brazil, Solmaz Abdi, Aneesha Bakharia, Shazia Sadiq, Dragan Gašević, Paul Denny, and Hassan Khosravi. 2024. Large language models meet user interfaces: The case of provisioning feedback. *Computers and Education: Artificial Intelligence* 7 (2024), 100289. doi:10.1016/j.caeai.2024.100289

[76] James Prather, Paul Denny, Brett A. Becker, Robert Nix, Brent N. Reeves, Arisoa S. Randrianasolo, and Garrett Powell. 2023. First Steps Towards Predicting the Readability of Programming Error Messages. In *Proceedings of the 54th ACM Technical Symposium on Computer Science Education V. 1* (Toronto ON, Canada) *(SIGCSE 2023)*. Association for Computing Machinery, New York, NY, USA, 549–555. doi:10.1145/3545945.3569791

[77] James Prather, Raymond Pettit, Kayla McMurry, Alani Peters, John Homer, and Maxine Cohen. 2018. Metacognitive Difficulties Faced by Novice Programmers in Automated Assessment Tools. In *Proceedings of the 2018 ACM Conference on International Computing Education Research* (Espoo, Finland) *(ICER '18)*. Association for Computing Machinery, New York, NY, USA, 41–50. doi:10.1145/3230977.3230981

[78] James Prather, Raymond Pettit, Kayla Holcomb McMurry, Alani Peters, John Homer, Nevan Simone, and Maxine Cohen. 2017. On Novices' Interaction with Compiler Error Messages: A Human Factors Approach. In *Proceedings of the 2017 ACM Conference on International Computing Education Research* (Tacoma, Washington, USA) *(ICER '17)*. Association for Computing Machinery, New York, NY, USA, 74–82. doi:10.1145/3105726.3106169

[79] James Prather, Brent N. Reeves, Paul Denny, Brett A. Becker, Juho Leinonen, Andrew Luxton-Reilly, Garrett Powell, James Finnie-Ansley, and Eddie Antonio Santos. 2023. "It's Weird That it Knows What I Want": Usability and Interactions with Copilot for Novice Programmers. *ACM Trans. Comput.-Hum. Interact.* 31, 1, Article 4 (Nov. 2023), 31 pages. doi:10.1145/3617367

[80] James Prather, Brent N Reeves, Juho Leinonen, Stephen MacNeil, Arisoa S Randrianasolo, Brett A. Becker, Bailey Kimmel, Jared Wright, and Ben Briggs. 2024. The Widening Gap: The Benefits and Harms of Generative AI for Novice Programmers. In *Proceedings of the 2024 ACM Conference on International Computing Education Research - Volume 1* (Melbourne, VIC, Australia) *(ICER '24)*. Association for Computing Machinery, New York, NY, USA, 469–486. doi:10.1145/3632620.3671116

[81] Thomas W. Price, Samiha Marwan, and Joseph Jay Williams. 2021. Exploring Design Choices in Data-Driven Hints for Python Programming Homework. In *Proceedings of the Eighth ACM Conference on Learning @ Scale* (Virtual Event, Germany) *(L@S '21)*. Association for Computing Machinery, New York, NY, USA, 283–286. doi:10.1145/3430895.3460159

[82] Thomas W. Price, Samiha Marwan, Michael Winters, and Joseph Jay Williams. 2020. An Evaluation of Data-Driven Programming Hints in a Classroom Setting. In *Artificial Intelligence in Education*, Ig Ibert Bittencourt, Mutlu Cukurova, Kasia Muldner, Rose Luckin, and Eva Millán (Eds.). Springer International Publishing, Cham, 246–251.

[83] Arun Raman and Viraj Kumar. 2022. Programming Pedagogy and Assessment in the Era of AI/ML: A Position Paper. In *Proceedings of the 15th Annual ACM India Compute Conference* (Jaipur, India) *(COMPUTE '22)*. Association for Computing Machinery, New York, NY, USA, 29–34. doi:10.1145/3561833.3561843

[84] Eric F Rietzschel, Bernard A Nijstad, and Wolfgang Stroebe. 2006. Productivity is not enough: A comparison of interactive and nominal brainstorming groups on idea generation and selection. *Journal of Experimental Social Psychology* 42, 2 (2006), 244–251.

[85] Kelly Rivers. 2017. *Automated data-driven hint generation for learning programming*. Ph. D. Dissertation. Carnegie Mellon University.

[86] Lianne Roest, Hieke Keuning, and Johan Jeuring. 2024. Next-Step Hint Generation for Introductory Programming Using Large Language Models. In *Proceedings of the 26th Australasian Computing Education Conference* (Sydney, NSW, Australia) *(ACE '24)*. Association for Computing Machinery, New York, NY, USA, 144–153. doi:10.1145/3636243.3636259

[87] Vincent Donche San Verhavert, Renske Bouwer and Sven De Maeyer. 2019. A meta-analysis on the reliability of comparative judgement. *Assessment in Education: Principles, Policy & Practice* 26, 5 (2019), 541–562. doi:10.1080/0969594X.2019.1602027 arXiv:https://doi.org/10.1080/0969594X.2019.1602027

[88] Andreas Scholl and Natalie Kiesler. 2024. How Novice Programmers Use and Experience ChatGPT when Solving Programming Exercises in an Introductory Course. arXiv:2407.20792 [cs.AI] https://arxiv.org/abs/2407.20792

[89] Judy Sheard, Paul Denny, Arto Hellas, Juho Leinonen, Lauri Malmi, and Simon. 2024. Instructor Perceptions of AI Code Generation Tools - A Multi-Institutional Interview Study. In *Proceedings of the 55th ACM Technical Symposium on Computer Science Education V. 1* (Portland, OR, USA) *(SIGCSE 2024)*. Association for Computing Machinery, New York, NY, USA, 1223–1229. doi:10.1145/3626252.3630880

[90] Brad Sheese, Mark Liffiton, Jaromir Savelka, and Paul Denny. 2024. Patterns of Student Help-Seeking When Using a Large Language Model-Powered Programming Assistant. In *Proceedings of the 26th Australasian Computing Education Conference* (Sydney, NSW, Australia) *(ACE '24)*. Association for Computing Machinery, New York, NY, USA, 49–57. doi:10.1145/3636243.3636249

[91] Valerie J. Shute. 2008. Focus on Formative Feedback. *Review of Educational Research* 78, 1 (2008), 153–189. doi:10.3102/0034654307313795 arXiv:https://doi.org/10.3102/0034654307313795

[92] Rebecca Smith and Scott Rixner. 2019. The Error Landscape: Characterizing the Mistakes of Novice Programmers. In *Proceedings of the 50th ACM Technical Symposium on Computer Science Education* (Minneapolis, MN, USA) *(SIGCSE '19)*. Association for Computing Machinery, New York, NY, USA, 538–544. doi:10.1145/3287324.3287394

[93] Ryo Suzuki, Gustavo Soares, Elena Glassman, Andrew Head, Loris D'Antoni, and Björn Hartmann. 2017. Exploring the Design Space of Automatically Synthesized Hints for Introductory Programming Assignments. In *Proceedings of the 2017 CHI Conference Extended Abstracts on Human Factors in Computing Systems* (Denver, Colorado, USA) *(CHI EA '17)*. Association for Computing Machinery, New York, NY, USA, 2951–2958. doi:10.1145/3027063.3053187

[94] Jacqueline Whalley, Amber Settle, and Andrew Luxton-Reilly. 2023. A Think-Aloud Study of Novice Debugging. *ACM Trans. Comput. Educ.* 23, 2, Article 28 (June 2023), 38 pages. doi:10.1145/3589004

[95] Joseph B. Wiggins, Fahmid M. Fahid, Andrew Emerson, Madeline Hinckle, Andy Smith, Kristy Elizabeth Boyer, Bradford Mott, Eric Wiebe, and James Lester. 2021. Exploring Novice Programmers' Hint Requests in an Intelligent Block-Based Coding Environment. In *Proceedings of the 52nd ACM Technical Symposium on Computer Science Education* (Virtual Event, USA) *(SIGCSE '21)*. Association for Computing Machinery, New York, NY, USA, 52–58. doi:10.1145/3408877.3432538

[96] Marvin N. Wright and Andreas Ziegler. 2017. ranger: A Fast Implementation of Random Forests for High Dimensional Data in C++ and R. *Journal of Statistical Software* 77, 1 (2017), 1–17. doi:10.18637/jss.v077.i01

[97] Ruiwei Xiao, Xinying Hou, and John Stamper. 2024. Exploring How Multiple Levels of GPT-Generated Programming Hints Support or Disappoint Novices. In *Extended Abstracts of the CHI Conference on Human Factors in Computing Systems* (Honolulu, HI, USA) *(CHI EA '24)*. Association for Computing Machinery, New York, NY, USA, Article 142, 10 pages. doi:10.1145/3613905.3650937

[98] 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 *Proceedings of the 46th International Conference on Software Engineering: Software Engineering Education and Training* (Lisbon, Portugal) *(ICSE-SEET '24)*. Association for Computing Machinery, New York, NY, USA, 331–341. doi:10.1145/3639474.3640076

[99] J.D. Zamfirescu-Pereira, Richmond Y. Wong, Bjoern Hartmann, and Qian Yang. 2023. Why Johnny Can't Prompt: How Non-AI Experts Try (and Fail) to Design LLM Prompts. In *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems* (Hamburg, Germany) *(CHI '23)*. Association for Computing Machinery, New York, NY, USA, Article 437, 21 pages. doi:10.1145/3544548.3581388