Prompting Large Language Models to Power Educational Chatbots

Juan Carlos Farah[1,2](✉), Sandy Ingram[2], Basile Spaenlehauer[1], Fanny Kim-Lan Lasne[1], and Denis Gillet[1]

[1] École Polytechnique Fédérale de Lausanne (EPFL), Lausanne, Switzerland
{juancarlos.farah,basile.spaenlehauer,fanny.lasne,denis.gillet}@epfl.ch
[2] University of Applied Sciences (HES-SO), Fribourg, Switzerland
{juancarlos.farah,sandy.ingram}@hefr.ch

Abstract. The recent rise in both popularity and performance of large language models has garnered considerable interest regarding their applicability to education. Technologies like ChatGPT, which can engage in human-like dialog, have already disrupted educational practices given their ability to answer a wide array of questions. Nevertheless, integrating these technologies into learning contexts faces both technological and pedagogical challenges, such as providing appropriate user interfaces and configuring interactions to ensure that conversations stay on topic. To better understand the potential large language models have to power educational chatbots, we propose an architecture to support educational chatbots that can be powered by these models. Using this architecture, we created a chatbot interface that was integrated into a web application aimed at teaching software engineering best practices. The application was then used to conduct a case study comprising a controlled experiment with 26 university software engineering students. Half of the students interacted with a version of the application equipped with the chatbot, while the other half completed the same lesson without the chatbot. While the results of our quantitative analysis did not identify significant differences between conditions, qualitative insights suggest that learners appreciated the chatbot. These results could serve as a starting point to optimize strategies for integrating large language models into pedagogical scenarios.

Keywords: Educational Chatbots · Prompting · GPT-3 · Large Language Models · Software Engineering Education · Digital Education

1 Introduction

As large language models (LLMs) become more accessible through publicly available application programming interfaces (APIs), the potential these models have to support a wide variety of pedagogical scenarios is becoming evident. One obvious application of these models is to power educational chatbots. Indeed, researchers have advocated for the use of LLMs to support more natural conversation and overcome the limitations of rule-based systems or systems based on limited training data [15]. However, several challenges limit the integration of LLM-powered chatbots into educational contexts. These challenges include—among others—providing the appropriate user interfaces and configuring the LLMs to ensure that the generated text is aligned with the pedagogical scenario in which the educational chatbots are deployed [11].

To help address these challenges, we designed an architecture that developers in education can follow to create interfaces that educators can use to configure and deploy LLM-powered chatbots for use in digital education. At its core, this architecture considers chatbots as configuration objects that define the user interface elements that will be shown to the learner as well as the prompt that will be sent to the LLM. Following this architecture, we created a chatbot interface that was embedded in a web application designed to execute and review snippets of code. We used this application in a case study consisting of a between-subjects controlled experiment with 26 software engineering students. While 13 students completed a lesson on programming best practices supported by the chatbot, the other 13 students completed the same lesson without support from the chatbot. Our mixed-method analysis of the data from this experiment focused on how the chatbot affected five aspects of the learning experience. A sixth aspect—*conversation*—was applicable only to the treatment group that was exposed to the chatbot.

Given the increasing interest in the applications of powerful natural language processing (NLP) technologies, our study is timely and relevant to both research and practice. Our architecture can serve developers in education looking to integrate chatbots into digital education platforms, while findings from our case study can guide researchers and educators in conducting future empirical studies and incorporating educational chatbots into their practice.

2 Background and Related Work

Our research is underpinned by advances in NLP methods and, in particular, by the success of LLMs. LLMs are used for a wide array of language tasks and can also be used to power chatbots. The most notable example is OpenAI's *ChatGPT* [13]. Without further configuration, this open-domain chatbot can interact with users on diverse topics through unrestrained conversations. However, understanding how LLMs can be harnessed to create *task-oriented* chatbots for specific domains is still the focus of ongoing research.

A recent position paper by Kasneci *et al.* [11] outlined opportunities and challenges of incorporating LLMs into education. Among the challenges identified, the authors highlight (i) the possible biases that LLMs can perpetuate and amplify, (ii) the need for open educational resources (OERs) to guide educators on how to access and use these models, (iii) the need to ensure data privacy and security, and (iv) the lack of adaptability to align the models with the objectives of individual learners and educators. Addressing these challenges could open the door to more powerful applications of LLMs to education. Kasneci *et al.* also underlined the need to ensure that the interfaces used to interact with these models are aligned with the needs of different types of learners (e.g., adapting for age-related constraints and accessibility requirements).

An important factor to consider when assessing the applicability of these LLMs to education is the process through which they can be configured for specific tasks, such as generating the responses that an educational chatbot can use in its interactions. This process is supported by *prompting*, which consists in providing the model with instructions and a few examples of how it should respond to a query. Finding the most appropriate prompts is a complex task and using inefficient prompting strategies can result in worse results than using no prompt at all [14]. Identifying optimal prompts—or *prompt engineering*—is an active area of research in NLP. Recent work has focused on automating the generation of these prompts [9], understanding the biases that prompting can be susceptible to [18], and providing the appropriate infrastructure to generate prompts [2]. Nevertheless, there is little guidance for performing prompt engineering with domain-specific applications in mind, as is the case with educational chatbots.

These gaps in the research motivate the design of our architecture, which aims to help developers in education provide interfaces for educators to easily configure and deploy chatbots within digital education platforms.

3 Design

Our architecture—depicted in Fig. 1—consists of six components that interact through three main processes. These components are loosely coupled and abstractly defined in order to allow developers to adapt them as needed for the particular constraints that might be present in different digital education platforms. In this section, we detail the functionalities provided by each component and the processes they support.

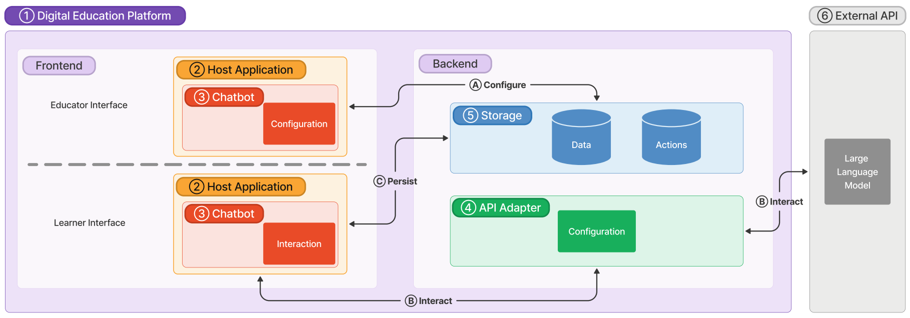


Fig. 1. Our design comprises six loosely coupled components (labeled 1–6) that interact through three main processes (labeled A–C).

3.1 Components

The architecture defines six components that interact to facilitate the integration of chatbots powered by LLMs into educational contexts: (1) digital education platform, (2) host application, (3) chatbot, (4) API adapter, (5) storage, (6) external API.

1. **Digital Education Platform:** The learning activity takes place on a digital education platform. This platform should provide a way to differentiate content that is visible to educators from content that is visible to learners. Using a dedicated view, educators should be able to create a learning activity by curating multimedia content, such as images, text, and videos. Most importantly, educators should be able to select interactive learning resources (e.g., web applications) that can host chatbots. That is, chatbots are not directly embedded in the digital education platform, but in interactive applications that can be added to the learning activity. We refer to these applications as *host applications*.
2. **Host Application:** The host application is an interactive learning resource that can be embedded in digital education platforms to support learner interactions with chatbots. A host application exposes a configuration panel for educators to activate and configure chatbots. Once activated, these chatbots are visible to learners interacting with the application. For this, the host application should provide an interface through which learners can communicate with the chatbot (e.g., a forum, chat box, or message thread). How a chatbot interacts with learners is entirely defined by its configuration and the interaction affordances provided by the host application.
3. **Chatbot:** A chatbot in our architecture is not an application by default, but a component that can be activated and configured *within* an application. This increases the portability and customizability of a chatbot, which can be developed independently from the host application(s) it can be embedded in, and by educators without technical backgrounds. Hence, the chatbot can be considered to be an OER that is defined by its configuration. This configuration consists of a name, an avatar, a scope (i.e., who can see the messages sent by individual students), a prompt (which is sent to the LLM), and a cue (i.e., a message displayed to learners to invite them to interact).
4. **API Adapter:** The API adapter is embedded in the digital education platform and serves to communicate with the API exposed by the LLM provider. This adapter can also be deployed outside of a single digital education platform, thus serving multiple platforms. Given that adapters interface with APIs external to the digital education platform, they can also provide fallback responses in case the APIs are not accessible. Nevertheless, adapters are stateless and should only serve to handle requests from host applications, thus delegating the storage of any information to the digital education platform.
5. **Storage:** Two types of data are stored by the digital education platform: (i) application data and (ii) application actions. Application data refer to the content of a learner's interaction with the chatbot, including both the

messages provided by the learner and the chatbot's responses. Application actions refer to activity traces of how a learner interacts with the conversational interface in which the chatbot is embedded, including keystrokes, clicks, and other events that can be captured by the browser. These actions can serve to deliver learning analytics and provide a more nuanced depiction of how learners interact with chatbots.

6. **External API:** Finally, our architecture requires an LLM that is accessible through a public API in order to generate the responses to the queries provided by the learners.

3.2 Process

There are three main processes that define the interactions between the aforementioned components. These processes are outlined below.

(A) **Configure:** Educators should be able to configure the chatbot through the educator interface of the host application. The chatbot's configuration is kept in the storage component and allows for the personalization of the chatbot integration for each educator's particular needs. This configuration is also seamlessly reproducible, allowing educators to quickly replicate chatbot integrations across learning activities.

(B) **Interact:** The core process defined by our architecture is how learners interact with the chatbot. When a learner interacts with the chatbot, the host application's learner interface connects to an external LLM API provider through the API adapter component hosted in the digital education platform's backend.

(C) **Persist:** As interactions take place, the host application ensures that the outcomes of these interactions are *persisted* on the digital education platform. These outcomes mainly concern the conversation between the chatbot and the learner (and any related content such as emoji reactions) but also include any actions that the learner might take using the chatbot interface. These actions can then be used to provide learning analytics.

4 Methodology

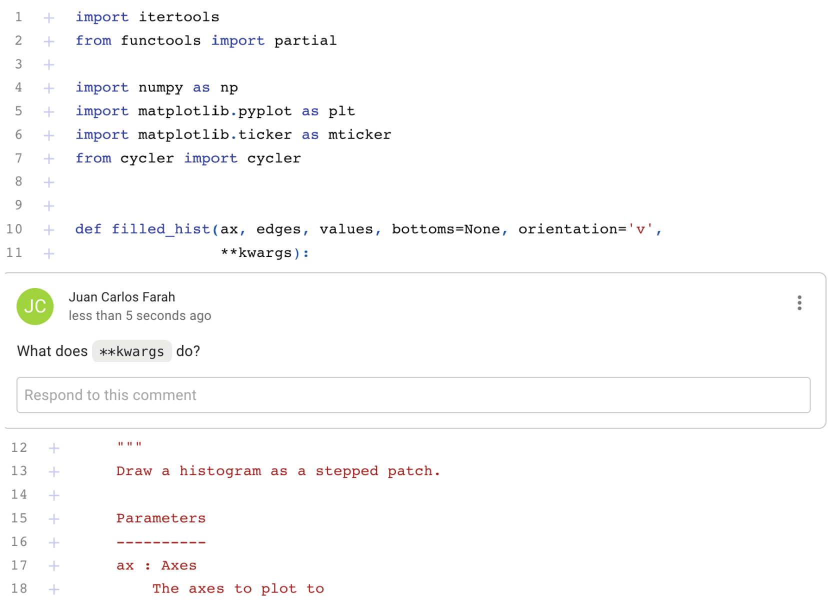
To test the applicability of our architecture in practice, we implemented a chatbot integration following our design and conducted an evaluation focused on addressing one main research question:

> *How does incorporating a large language model-based chatbot to support a lesson on software engineering best practices affect the learning experience?*

We conducted this study in January 2023. This evaluation took the form of a case study consisting of a between-subjects controlled experiment comprising one control (no chatbot) and one treatment (chatbot) condition. For both conditions, we analyzed five aspects of the learning experience: (i) *short-term learning* *gains*, (ii) *engagement*, (iii) *self-reflection on the learning experience*, (iv) *feedback regarding the lesson*, and (v) *usability*. For the treatment group, we also analyzed a sixth aspect (vi) *conversation*, which focused on the exchanges the learners had with the chatbot. The small-scale nature of our study allowed us to conduct a mixed-method analysis. In this section, we present our methodology.

4.1 Scenario

To ensure ecological validity, our evaluation took place in a formal education setting. As part of their coursework, students completed an in-class online lesson consisting of an ungraded 45-min exercise. The exercise comprised a *code review notebook* [5] covering JavaScript code style standards. Code review notebooks allow educators to scaffold pedagogical scenarios that introduce the code review process to learners through code snippets, following a template resembling computational notebooks. More specifically, we started by introducing the concept of *linting* code, which involves the use of static analysis tools to detect issues in software [10]. The lesson then covered ESLint [17], a linter for JavaScript [16], as well as the Airbnb JavaScript Style Guide [1], a popular configuration for ESLint. In this section, we outline the technological context and pedagogical scenario used in our evaluation.



```
import itertools
from functools import partial

import numpy as np
import matplotlib.pyplot as plt
import matplotlib.ticker as mticker
from cycler import cycler


def filled_hist(ax, edges, values, bottoms=None, orientation='v',
                **kwargs):
```

Juan Carlos Farah
less than 5 seconds ago

What does `**kwargs` do?

Respond to this comment

```
    """
    Draw a histogram as a stepped patch.

    Parameters
    ----------
    ax : Axes
        The axes to plot to
```

Fig. 2. The Code Capsule application can be used to write, execute, and review code. The code used in this example has been adapted from the Python Matplotlib library's documentation [7].

Technological Context. In this section, we describe how we implemented our architecture to provide the technological context for our experiment. Our chatbot was integrated into the *Graasp* digital education platform [6] through *Code Capsule* as the host application. Code Capsule is an application that allows learners to both review and execute code (see Fig. 2). This application supports the integration of chatbots when used to review code.

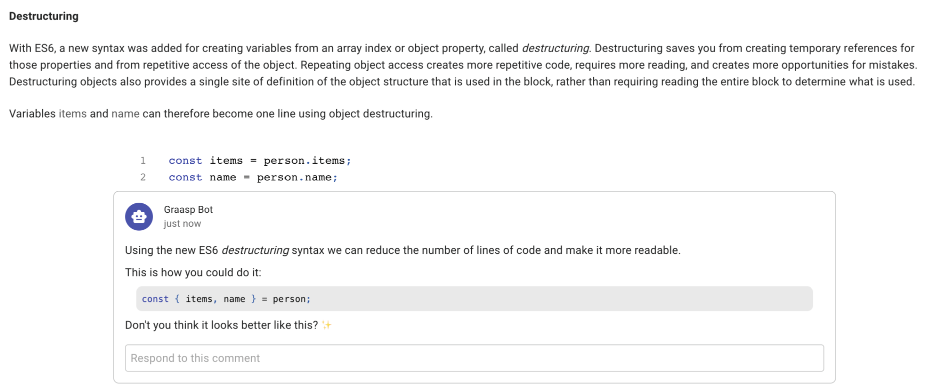
The chatbot that was integrated into Code Capsule was configured as follows. To remain gender-neutral and maximize consistency with Graasp, we named our chatbot *Graasp Bot* and represented it with a robot avatar. All chatbot interactions were scoped at the individual level, so students could only see their own interactions and not those of their peers. The prompts for each code snippet were only visible to the educator via Code Capsule's configuration panel and were prepended to the conversation as learners interacted with the chatbot. These prompts were defined following the pattern below[1]:

> *The following is a conversation between a chatbot and a student discussing the correction of an exercise about linting, ESLint, code styling, and best practices in JavaScript. After each response, the chatbot gives the student one or two options to continue the conversation.*
>
> ***Chatbot:***
> *`<EXPLANATION OF WHAT IS WRONG IN THE CODE SNIPPET>`. So, I would change the following line of code:*
>
> *`<INCORRECT CODE>`*
>
> *To the following:*
>
> *`<CORRECT CODE>`*
>
> *`<QUESTION ASKING THE STUDENT IF THEY UNDERSTAND THE DIFFERENCE>`*
>
> ***Student:*** OK. `<QUESTION ASKING THE CHATBOT SOMETHING RELATED TO THE ISSUE>`
>
> ***Chatbot:*** `<ANSWER>`. Do you want me to provide you with an example of *`<THE ISSUE>`*?
>
> ***Student:*** No, that's OK. Thanks! Any other issues I should be aware of?

To invite users to interact with the chatbot, we showed them a message that was embedded as a comment in the code snippet featured on Code Capsule. We refer to these messages as *cues*. Cues were dependent on each code snippet in

[1] Note that placeholders are presented between angle brackets (`<>`).

which the chatbot was embedded. All of these cues followed the same pattern (see Fig. 3). This pattern consisted of (i) an explanation of an issue present in the code snippet, (ii) a suggestion on how to fix the issue, and (iii) a question asking students whether they agreed with some facet of the proposed solution.


Destructuring

With ES6, a new syntax was added for creating variables from an array index or object property, called *destructuring*. Destructuring saves you from creating temporary references for those properties and from repetitive access of the object. Repeating object access creates more repetitive code, requires more reading, and creates more opportunities for mistakes. Destructuring objects also provides a single site of definition of the object structure that is used in the block, rather than requiring reading the entire block to determine what is used.

Variables items and name can therefore become one line using object destructuring.

```
const items = person.items;
const name = person.name;
```

Graasp Bot
just now

Using the new ES6 *destructuring* syntax we can reduce the number of lines of code and make it more readable.

This is how you could do it:

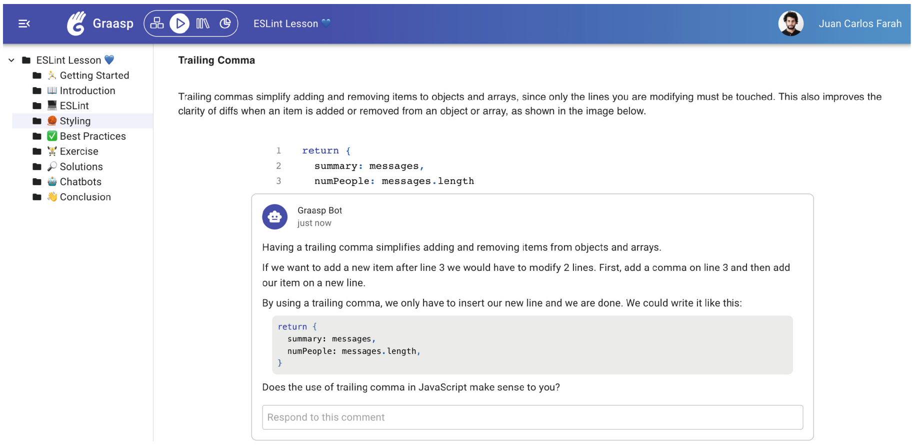
```
const { items, name } = person;
```

Don't you think it looks better like this? ✨

Fig. 3. The cue consisted of an explanation of the issue present in the snippet, a suggested fix, and a question asking students whether they agreed with the solution proposed.

When students interacted with our chatbot, Code Capsule interfaced with the external API exposed by OpenAI through an API adapter that was integrated into Graasp. The adapter was configured to use the GPT-3 `text-davinci-003` model, with a `temperature` parameter of 0.9 (higher values make completions of the same prompt more random), a `presence_penalty` of 0.6 (higher values penalize new tokens if they have already appeared), a `top_p` parameter of 1 (always returning the best completion), and the maximum number of tokens to be included in the chatbot response fixed at 150.

Pedagogical Scenario. The lesson used in this experiment followed the *Fixer Upper* pedagogical pattern [3]. For coding exercises, this pedagogical pattern consists of presenting students with code "that is generally sound but [contains] carefully introduced flaws [that] can both introduce a complex topic early and serve as a way to introduce error analysis and correction" [3]. In our case, we structured the lesson over 10 phases, which students were meant to navigate sequentially (see Fig. 4). Students were first introduced to the lesson (Phase 1) and then asked to complete a short exercise that served as a pre-test to gauge their knowledge of JavaScript code style standards (Phase 2). Phases 3 (*Introduction*) and 4 (*ESLint*) covered the concept of code linting and ESLint specifically, while Phases 5 (*Styling*) and 6 (*Best Practices*) presented examples of code style standards that students should follow when writing JavaScript code. In these phases, 10 code snippets were included using Code Capsule alongside a textual explanation of the issue present in the snippet. For students in the treatment condition, the code snippet also included a cue from Graasp Bot, as shown in Fig. 3. Phase 7 (*Exercise*) consisted of an exercise that served as a post-test, while Phase 8 (*Solutions*) presented the solutions to the exercise and asked the students to reflect on their performance in the exercise. For students in the treatment condition, Phase 8 also included explanations presented by Graasp Bot. Phase 9 (*Chatbots*) was also different between the control and treatment conditions. On the one hand, students in the control condition were asked to imagine how they would integrate chatbots into the exercise and to provide sample dialogs that they would envision having with the chatbot. On the other hand, students in the treatment condition were asked to report on their experience interacting with the chatbot. Finally, Phase 10 (*Conclusion*) served as a conclusion to the exercise and provided a link to a questionnaire.



Trailing Comma

Trailing commas simplify adding and removing items to objects and arrays, since only the lines you are modifying must be touched. This also improves the clarity of diffs when an item is added or removed from an object or array, as shown in the image below.

```
1  return {
2    summary: messages,
3    numPeople: messages.length
```

Graasp Bot
just now

Having a trailing comma simplifies adding and removing items from objects and arrays.

If we want to add a new item after line 3 we would have to modify 2 lines. First, add a comma on line 3 and then add our item on a new line.

By using a trailing comma, we only have to insert our new line and we are done. We could write it like this:

```
return {
  summary: messages,
  numPeople: messages.length,
}
```

Does the use of trailing comma in JavaScript make sense to you?

Respond to this comment

Fig. 4. The lesson used in this study consisted of a code review notebook aimed at teaching software engineering best practices and included the ten phases shown in the sidebar. In this figure, we highlight the *Styling* phase and show the explanation block—including a cue from the chatbot—recommending the use of trailing commas.

4.2 Participants

We recruited 28 third-year bachelor students taking part in a course on human-computer interaction at the School of Engineering and Architecture of Fribourg, Switzerland. A total of 26 students—25 male, 1 female—completed the study. Students were informed that this was an ungraded, optional exercise. To encourage participation, students were given extra credit for completing the activity.

4.3 Instruments

Short-term learning gains were operationalized based on the learner's performance in the pre- and post-tests by calculating the difference between both tests. These gains could range from -100% to 100%. Engagement was measured by calculating the total amount of time spent in each phase during the one-hour time frame that was allocated for the lesson. Self-reflection and feedback were respectively operationalized through the following two open-ended questions: (i) *Did you manage to find all of these [issues]? If not, which ones did you miss? Did you find any of them particularly tricky/helpful?* and (ii) *What did you think about this lesson? Any comments, suggestions, or feedback?*. A second feedback question was relevant only to the students in the treatment group: *In a few phrases, describe your experience interacting with the chatbot used in this activity. What did you like about it? What could be improved?* This second question was only used for our qualitative analysis of the feedback aspect. Usability was measured with the User Experience Questionnaire (UEQ), a standard instrument that measures usability across six dimensions [12]. Finally, for students in the treatment group, our analysis included the conversations students had with the chatbot. This qualitative analysis focused on (i) whether conversations were on topic, (ii) how long the conversations were, (iii) how natural the conversations were, and (iv) what responses were elicited by the different types of cues.

4.4 Data Analysis

We applied both descriptive and inferential statistics to our quantitative data, reporting the means ($\bar{x}$), medians ($\tilde{x}$), standard deviations (s_x), minima ($x_{\min}$), and maxima ($x_{\max}$), as well as the results of t-tests for independent samples comparing across the two conditions. To perform sentiment analyses on students' self-reflection and feedback responses, we used VADER [8], which assigns a sentiment score ranging from -1 (negative sentiment) to $+1$ (positive sentiment). The results of the UEQ were analyzed using its data analysis toolkit [12]. Finally, open-ended responses were analyzed using qualitative methods, following line-by-line data coding [4].

5 Results

In this section, we present our results with respect to the aspects studied.

5.1 Learning Gains

The mean learning gains were $\bar{x} = 0.429$ ($\tilde{x} = 0.369$, $s_x = 0.212$, $x_{\min} = 0.115$, $x_{\max} = 0.746$) in the control condition and $\bar{x} = 0.430$ ($\tilde{x} = 0.492$, $s_x = 0.243$, $x_{\min} = -0.146$, $x_{\max} = 0.692$) in the treatment condition. These results—illustrated in Fig. 5—show that both conditions led to, on average, positive learning gains for students, with all students except one achieving positive learning gains. While the median learning gain was higher in the treatment condition, a t-test did not show a significant difference between conditions.

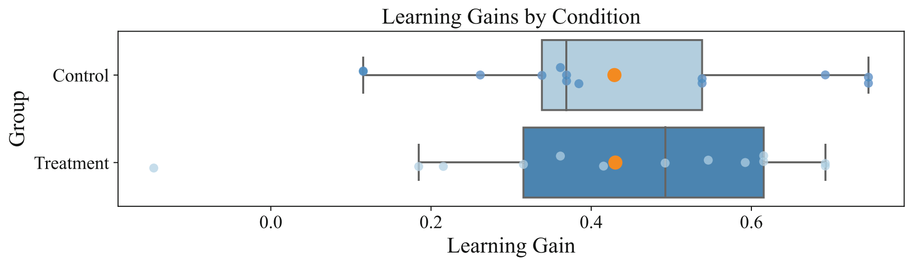


Fig. 5. Learning Gains

5.2 Engagement

On average, learners in the treatment condition spent a total of $\bar{x} = 41.7$ min ($\tilde{x} = 37.8$, $s_x = 9.31$, $x_{\min} = 29.4$, $x_{\max} = 59.5$) to complete the lesson, while learners in the control condition did so in $\bar{x} = 40.1$ min ($\tilde{x} = 36.5$, $s_x = 10.4$, $x_{\min} = 22.2$, $x_{\max} = 55.2$). There were no significant differences in the time spent by students either overall (see Fig. 6) or across the 10 phases that constituted our lesson (see Fig. 7).

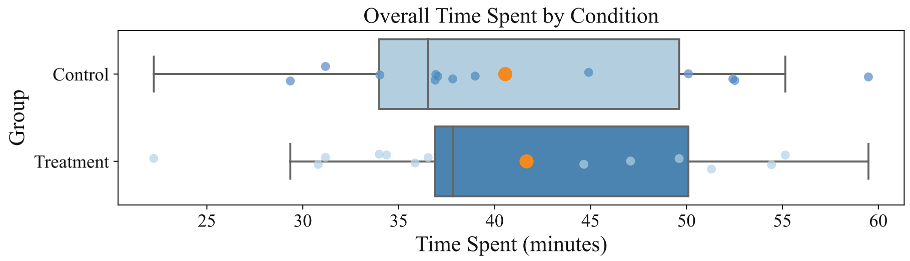


Fig. 6. Engagement

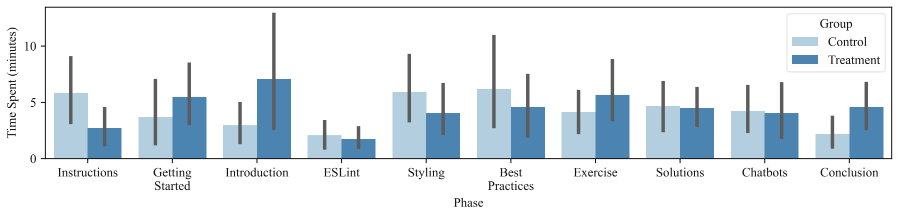


Fig. 7. Time Spent per Phase

5.3 Self-reflection

Of the 26 students that participated in the study, 22 students (10 control, 12 treatment) provided responses to the self-reflection question. The sentiment analysis performed on student responses to the self-reflection question did not produce any significant differences between conditions. Nevertheless—as shown in Fig. 8—the distribution of scores in the treatment condition had a more positive tendency than the scores in the control condition. Specifically, the responses of the students in the control group resulted in a mean sentiment score of $\bar{x} = -0.128$ ($\tilde{x} = -0.0766$, $s_x = 0.454$, $x_{\min} = -0.743$, $x_{\max} = 0.757$), while those in the treatment condition resulted in a mean score of $\bar{x} = 0.0297$ ($\tilde{x} = 0.0386$, $s_x = 0.423$, $x_{\min} = -0.595$, $x_{\max} = 0.649$).

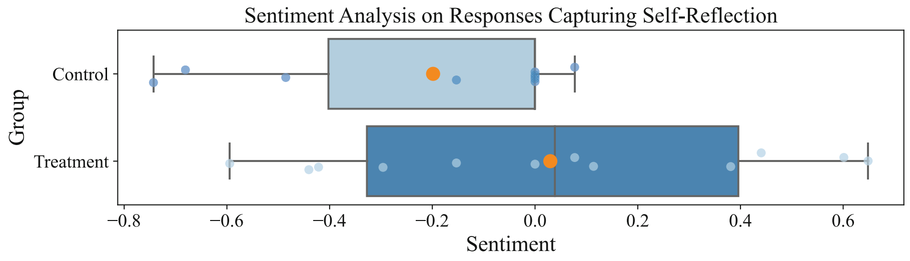


Fig. 8. Self-Reflection

Our qualitative analysis showed that responses were consistent between both groups, with students providing short answers in which they quickly described what they missed. All students, except five (two control, three treatment), specifically listed at least one issue they missed. A typical answer reads as follows: *"I forgot a `let` that had to be `const` , a comma after the last element of an object and a semicolon at the end of a line"*.

Six students (four control, two treatment) provided more detail regarding the issues they missed. In this case, a typical answer was: *"I forgot the first `let giftList` to `const giftList` despite seeing it for `total` at the end. I fell into the trap thinking it was redeclared in the `getGiftsTotal(person)` function because it had the same name"*.

5.4 Feedback

Of the 26 students that participated in the study, 22 students (10 control, 12 treatment) provided responses to the feedback question. The sentiment analysis performed on these responses did not yield any significant differences between conditions. As shown in Fig. 9, the distribution of the scores was mostly positive in both conditions, with only a few negative outliers. Responses from students in the control group resulted in a mean sentiment score of $\bar{x} = 0.451$ ($\tilde{x} = 0.556$, $s_x = 0.400$, $x_{\min} = -0.317$, $x_{\max} = 0.859$), while those in the treatment condition resulted in a mean score of $\bar{x} = 0.460$ ($\tilde{x} = 0.598$, $s_x = 0.331$, $x_{\min} = -0.356$, $x_{\max} = 0.796$).

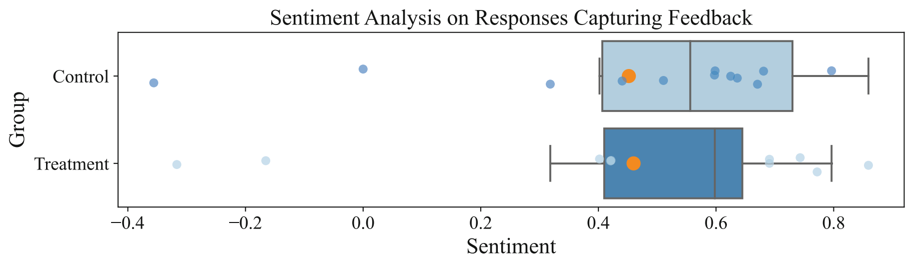


Fig. 9. Feedback

Our qualitative analysis of the first feedback question—which concerned both conditions—showed that all students except two (two treatment) provided a positive comment. Codes such as *good*, *great*, *fun*, *interesting*, and *helpful* were prevalent in students' responses under both conditions. Regarding the negative feedback, while one student provided a minor, off-topic comment regarding the color of the user interface, the other student questioned the need for a chatbot: *"Was the chatbot REALLY necessary? This lesson needs to end on a link to a lesson/tutorial on how to use/configure/install/firststeps/basics/... on ESLint"*.

Nevertheless, four students specifically provided positive feedback regarding their interactions with the chatbot, such as: *"It was helpful and I liked the interactivity with a bot"*. Finally, one student noted that while the interactivity and response time provided by the chatbot were positive, the chatbot would *never* replace the educator: *"ESLint was interesting. It's nice to have an answer directly to our questions. After that, it will never replace the answers of a teacher (even if the answer is direct)"*.

Furthermore, all students in the treatment group ($n = 13$) provided an answer to the second feedback question, which specifically asked about the chatbot and was therefore only visible to students in the treatment group. Although eight students provided positive feedback about the chatbot, four of these comments also included a note about how the chatbot had been *"repetitive"* or *"asked too many questions"*. One student noted the following:

> *"It was nice, but sometimes repetitive. He wished me twice a great day and when I asked how to implement something in the linter, it didn't show me the code. But overall, I find it ludic and it's a nice way to learn since we have interactions. May be interesting for children too. Maybe less with adults."*

Repetitiveness was also observed in the five negative comments, with students urging the chatbot to *"stop asking questions at the end"* or characterizing chatbots as *"pushy salesmen"*. One student provided the following constructive comment:

"Sometimes less interaction is more. In this lesson maybe too many interactions are offered and this could be at some point a bit annoying for the user. But still if correctly dosed it may bring some value for the user!"

5.5 Usability

Both groups rated the usability of the lesson positively. Compared to the UEQ benchmark, in the control group, the results achieved were above average (25% of results better, 50% of results worse) for four dimensions—*attractiveness*, *dependability*, *stimulation*, and *novelty*—while they were good (10% of results better, 75% of results worse) for *efficiency* and excellent (in the range of the 10% best results) for *perspicuity*. In the treatment group, the results achieved were above average for *stimulation*, good for three dimensions—*attractiveness*, *efficiency*, and *dependability*—and excellent for *perspicuity* and *novelty*.

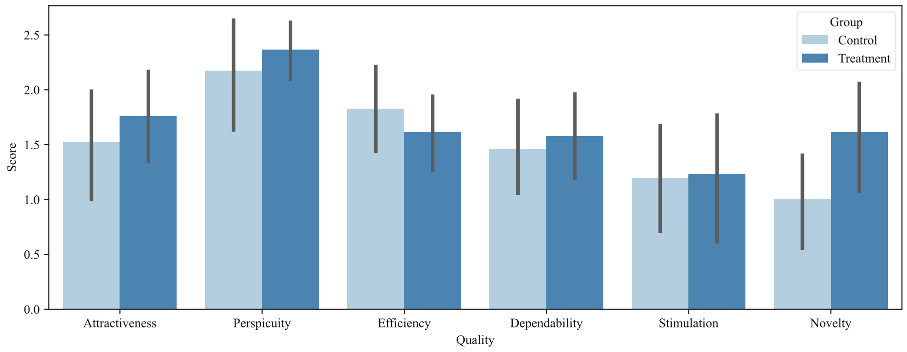


Fig. 10. Results of the User Experience Questionnaire (UEQ)

When comparing between conditions, however, two-sample t-tests did not result in any significant differences across any of the usability dimensions. Nevertheless, it is worth noting that the treatment condition achieved a better usability score in all dimensions, except *efficiency*, and specifically for the *novelty* dimension ($p = 0.0886$). These ratings are summarized in Fig. 10.

5.6 Conversation

For students in the treatment group, our analysis included a sixth aspect regarding the conversations students had with the chatbot. This analysis comprised 150 conversations.

First, all conversations were on topic. Occasionally students opened up the dialogue to cover broader subjects—such as JavaScript in general, instead of JavaScript code style—but these topics still fell within the lesson's scope.

Second—as shown in Fig. 11—approximately two-thirds of the conversations comprised three messages (only one message from the student and two from the

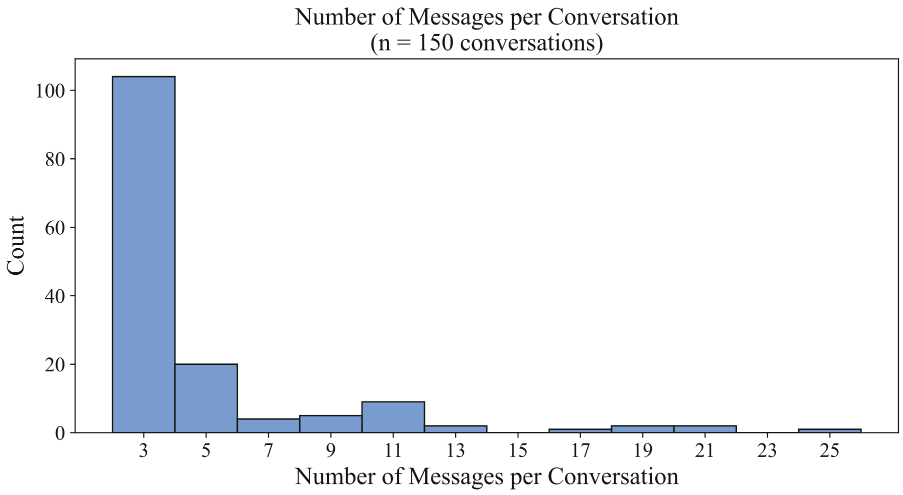


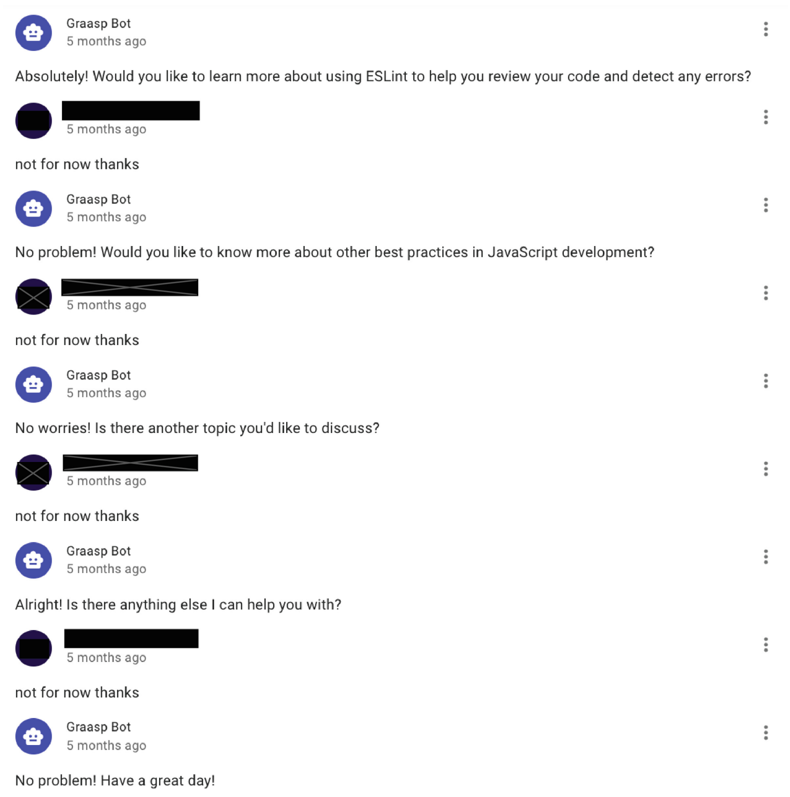
Fig. 11. Number of Messages per Conversation

chatbot). The other two-thirds of conversations were of length greater than or equal to five messages (i.e., at least two messages from the student).

Third, in most conversations, learners had to ask several times for the chatbot to stop asking if they had any further questions (see Fig. 12). Moreover—as mentioned above—in over two-thirds of the conversations, students did not pursue the exchange with the chatbot after one reply (as evidenced by the number of conversations comprising three messages). This means that students ignored the follow-up question from the chatbot, resulting in an unnatural end to the conversation.

Finally, we can divide the cues presented by the chatbot into three *types*: (i) those in which the student is invited to agree or disagree with its statements (*type I*, e.g., *Do you agree?*), (ii) those in which it asks for the student's opinion (*type II*, e.g., *What do you think about... ?*), and (iii) those coming after the exercises, in which it asks students whether they were able to identify a particular set of code style issues (*type III*, e.g., *Did you get all of these issues?*). Out of the 15 cues, seven were of type I, three were of type II, and five were of type III. We analyzed the format of the students' responses for each of these types. These can be short answers (e.g., "*yes*", "*no*", "*OK*") or long answers, i.e., developed and/or justified answers (e.g., "*It can be disturbing at first but it's readable too.*").

As shown in Fig. 13, approximately three-fourths of type I questions result in a short answer, after which most students leave the exchange (reflected in a conversation comprising three messages). On the other hand, we notice that about two-thirds of type II questions lead to long answers. Finally, for type III, we notice that the answer formats are almost equally distributed. By observing the content of these answers, we notice that short answers correspond most of the time to students who correctly identified the issues present in the exercise,


Graasp Bot
5 months ago

Absolutely! Would you like to learn more about using ESLint to help you review your code and detect any errors?

[redacted]
5 months ago

not for now thanks

Graasp Bot
5 months ago

No problem! Would you like to know more about other best practices in JavaScript development?

[redacted]
5 months ago

not for now thanks

Graasp Bot
5 months ago

No worries! Is there another topic you'd like to discuss?

[redacted]
5 months ago

not for now thanks

Graasp Bot
5 months ago

Alright! Is there anything else I can help you with?

[redacted]
5 months ago

not for now thanks

Graasp Bot
5 months ago

No problem! Have a great day!

Fig. 12. In many cases, the chatbot kept asking questions even though the learner sought to end the exchange, as shown by the repeated messages above.

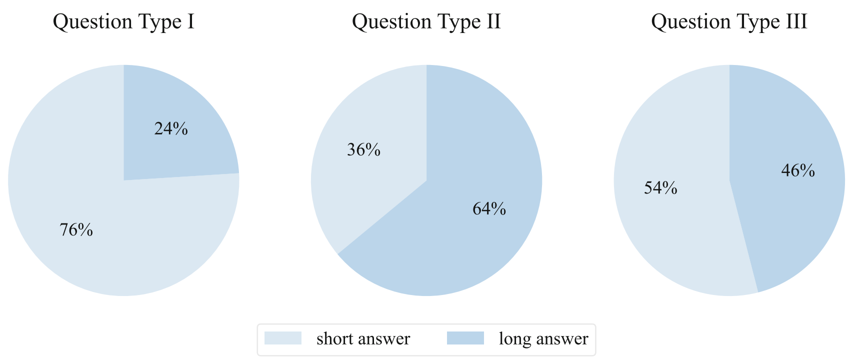


Fig. 13. Length of Answer by Type of Question

while long answers are those of students stating their mistakes and sometimes explaining the reasons for these mistakes or oversights.

6 Discussion

The results of our evaluation did not surface any significant differences between conditions for any of the five aspects considered for both conditions. However, there are a few points that stand out and provide interesting insights into the directions that we can explore in future work.

First, results were more positive in the treatment condition across all five aspects. That is, the mean learning gain was higher, students spent—on average—more time in the lesson, were more positive in reflecting on their performance in the post-test, provided more positive feedback, and rated the lesson higher on the UEQ. Although we emphasize that the differences were not statistically significant, these results offer a positive outlook for the integration of this type of educational chatbot into digital education platforms. At a minimum, these results show that educational chatbots following our architecture can complement pedagogical scenarios without interfering with the learning experience. Extensions of this study with larger cohorts, alternative instruments, and longer exposures could produce more concrete results.

Furthermore, qualitative feedback showed that, in general, learners appreciated the chatbot. Of the 12 students in the treatment condition who provided general feedback through a response to the first feedback question, five explicitly mentioned the chatbot, and only one did so to question whether its integration into the lesson was really necessary. Noting that the chatbot was intended to provide *extra* interaction in the lesson—in order not to compromise the learning experience of students in the control group—the chatbot was indeed designed to not *really* be necessary. However, in answers to the question that specifically asked students in the treatment condition about the chatbot, eight of the 13 students who responded provided a positive comment. The fact that the chatbot was appreciated by the majority of students who were exposed to it is a promising result.

Nevertheless, it is important to acknowledge that the repetitiveness of the questions posed by the chatbot negatively affected the learning experience, as made evident in the qualitative feedback provided by the students in the treatment group. These repeated questions were a result of the chatbot's configuration. More specifically, the chatbot's prompt (see Sect. 4.1) included the following instructions: *After each response, the chatbot gives the student one or two options to continue the conversation.* Although this strategy was effective in engaging students, it failed to capture the moment when the student wanted to stop interacting with the chatbot. Thus, this strategy quickly backfired and resulted in some students referring to the chatbot as *pushy* and *annoying*. This result sheds light on how what may appear to be a minor configuration detail could potentially have a negative impact on the user experience when using powerful LLMs.

Third, a close inspection of Fig. 7 shows that the average time spent in the phases that included chatbots (Styling, Best Practices, Solutions, and Chatbots) was actually longer for the control condition than for the treatment condition. While these differences are not significant, the consistency of these results across the four phases stands out. We would have expected students in the treatment condition to spend more time in these phases due to the extra interaction with the chatbot that students in the control condition—who only had to read the accompanying text—did not have to engage with. However, it could be the case that students in the treatment condition favored focusing on the explanation provided by the chatbot, which served as an interactive summary of what was contained in the text. Hence, the chatbot might have provided a faster way of learning the issue that was captured in each code snippet or simply a way to guide a student's focus through the exercise.

It is also worth pointing out that the results of the UEQ show that the differences between conditions were most significant for the *novelty* dimension. For this dimension, the ratings provided by students in the treatment condition are significantly higher than those provided by students in the control condition at the $p < 0.1$ level. The need for educational technologies to remain novel and attractive is particularly relevant in light of the rapidly changing technological landscape. Learning technologies and digital education platforms that achieve positive usability results in these dimensions are likely to have an advantage in attracting learners and keeping learners engaged.

With respect to the exchanges held between students in the treatment group and the chatbot, a first positive result is that conversations were all on topic. The fact that about a third of conversations consisted of five or more messages is an indicator that students took advantage of the opportunity to interact with the chatbot. However, conversations often lacked naturalness. As discussed above, this was caused by the fact that the chatbot was configured to end its messages with a question that was supposed to encourage the student to reply. This caused problems when the student wanted to end the conversation. Finally, the formulation of the cue appeared to have had an effect on whether student responses were short or long. While type I cues asking the student just to agree or disagree did not lead to long answers, type II cues asking the student to reflect were better at encouraging students to provide more in-depth answers.

7 Conclusion

In this paper, we presented the design of an architecture aimed at supporting developers in education in integrating LLM-powered chatbots into digital education platforms. We then conducted a case study comprising a between-subjects controlled experiment with 26 software engineering students. Half of the students completed the lesson with support from Graasp Bot—an educational chatbot embedded in an application implemented following our architecture—while the other half completed the lesson without support from the chatbot. Although there were no significant differences across the aspects considered in our evaluation, the results of our study can help optimize our prompt engineering strategy and provide useful examples for researchers and educators looking to incorporate LLM-powered chatbots into their practice. Furthermore, given that learning gains were not impacted by the presence of the chatbot, these findings reinforce the idea that educational chatbots could serve to provide additional information when educators are not available.

It is also important to note that there are a number of limitations that could have affected our findings. First, while our sample size was appropriate for our mixed methods experiment, expanding our study to include more subjects could help in the detection of differences between conditions, especially in pedagogical scenarios with shorter durations, where differences could be more subtle than expected. Similarly, increasing exposure by conducting semester-long or longitudinal studies could also serve to better identify the differences that emerge between conditions. Second, exploring this interaction strategy with different pedagogical scenarios and subject matter could help generalize the applicability of our architecture. Finally, incorporating standardized instruments to measure learning gains, self-reflection, and engagement could help reveal more interpretable results regarding how LLM-powered chatbots can provide support in educational contexts.

References

1. Airbnb: Airbnb JavaScript Style Guide (2012). https://airbnb.io/javascript/
2. Bach, S.H., et al.: PromptSource: An Integrated Development Environment and Repository for Natural Language Prompts (2022). https://doi.org/10.48550/arXiv.2202.01279
3. Bergin, J.: Fourteen pedagogical patterns. In: Devos, M., Rüping, A. (eds.) Proceedings of the 5th European Conference on Pattern Languages of Programs (EuroPLoP 2000). Universitaetsverlag Konstanz, Irsee, Germany (2000)
4. Charmaz, K.: Constructing Grounded Theory: A Practical Guide through Qualitative Analysis. Sage, London (2006)
5. Farah, J.C., Spaenlehauer, B., Rodríguez-Triana, M.J., Ingram, S., Gillet, D.: Toward code review notebooks. In: 2022 International Conference on Advanced Learning Technologies (ICALT), New York, NY, USA, pp. 209–211. IEEE (2022). https://doi.org/10.1109/ICALT55010.2022.00068
6. Gillet, D., Vonèche-Cardia, I., Farah, J.C., Phan Hoang, K.L., Rodríguez-Triana, M.J.: Integrated model for comprehensive digital education platforms. In: 2022 IEEE Global Engineering Education Conference (EDUCON), New York, NY, USA, pp. 1586–1592. IEEE (2022). https://doi.org/10.1109/EDUCON52537.2022.9766795
7. Hunter, J.D.: Matplotlib: a 2D graphics environment. Comput. Sci. Eng. **9**(3), 90–95 (2007)
8. Hutto, C.J., Gilbert, E.: VADER: a parsimonious rule-based model for sentiment analysis of social media text. In: Proceedings of the Eighth International AAAI Conference on Weblogs and Social Media, Ann Arbor, MI, USA, pp. 216–225. AAAI (2014). https://doi.org/10.1609/icwsm.v8i1.14550
9. Jiang, Z., Xu, F.F., Araki, J., Neubig, G.: How Can We Know What Language Models Know? (2020). https://doi.org/10.48550/arXiv.1911.12543

10. Johnson, S.C.: Lint, A C Program Checker. Technical report, Bell Laboratories, Murray Hill, NJ, USA (1978)
11. Kasneci, E., et al.: ChatGPT for Good? On Opportunities and Challenges of Large Language Models for Education (2023). https://doi.org/10.35542/osf.io/5er8f
12. Laugwitz, B., Held, T., Schrepp, M.: Construction and evaluation of a user experience questionnaire. In: Holzinger, A. (ed.) USAB 2008. LNCS, vol. 5298, pp. 63–76. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-89350-9_6
13. OpenAI: Introducing ChatGPT (2022). https://openai.com/blog/chatgpt
14. Reynolds, L., McDonell, K.: Prompt Programming for Large Language Models: Beyond the Few-Shot Paradigm (2021). https://doi.org/10.48550/arXiv.2102.07350
15. Song, D., Oh, E.Y., Rice, M.: Interacting with a conversational agent system for educational purposes in online courses. In: 2017 10th International Conference on Human System Interactions (HSI), Ulsan, South Korea, pp. 78–82. IEEE (2017). https://doi.org/10.1109/HSI.2017.8005002
16. Tómasdóttir, K.F., Aniche, M., van Deursen, A.: The adoption of JavaScript linters in practice: a case study on ESLint. IEEE Trans. Software Eng. **46**(8), 863–891 (2020). https://doi.org/10.1109/TSE.2018.2871058
17. Zakas, N.C.: ESLint (2013). https://eslint.org/
18. Zhao, T.Z., Wallace, E., Feng, S., Klein, D., Singh, S.: Calibrate Before Use: Improving Few-Shot Performance of Language Models (2021). https://doi.org/10.48550/arXiv.2102.09690