Supporting Students in Prototyping AI-backed Software with Hosted Prompt Template APIs

Timothy J. Aveni
tja@berkeley.edu
University of California, Berkeley
Berkeley, CA, USA

James Smith
james.smith@berkeley.edu
University of California, Berkeley
Berkeley, CA, USA

Armando Fox
fox@berkeley.edu
University of California, Berkeley
Berkeley, CA, USA

Björn Hartmann
bjoern@eecs.berkeley.edu
University of California, Berkeley
Berkeley, CA, USA

Abstract

Large AI models, such as LLMs and text-to-image models, can be used to power intelligent components of software applications. In our User Interface Design and Development course, we encourage students to practice integrating AI-powered capabilities into their software prototypes. To facilitate this practice, we developed REAGENT, an open source Web platform that facilitates student exploration through iterative authorship of prompt templates, then hosts AI model APIs for those templates with billing configured by instructors. After an introductory homework assignment, students were encouraged to invent their own AI features in an open-ended final team project. Students widely reported that their experiences with AI in this course were valuable, offering a deeper understanding of the capabilities and limitations of using AI in software. By supporting authorship, facilitating integration, providing visibility, and reducing administrative hurdles, REAGENT facilitated both student exploration and instructors' involvement. Additionally, we offer insight we gleaned as instructors into how students form conceptual models about AI.

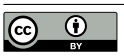
CCS Concepts

• **Human-centered computing** → **User interface programming**; **Natural language interfaces**; **Field studies**; • **Social and professional topics** → **Software engineering education**.

Keywords

generative AI, course technology, prototyping, large language models, text-to-image models, prompt engineering

ACM Reference Format:
Timothy J. Aveni, James Smith, Armando Fox, and Björn Hartmann. 2025. Supporting Students in Prototyping AI-backed Software with Hosted Prompt Template APIs. In *Proceedings of the 30th ACM Conference on Innovation and Technology in Computer Science Education V. 1 (ITiCSE 2025), June 27-July 2, 2025, Nijmegen, Netherlands.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3724363.3729109



ITiCSE 2025, June 27-July 2, 2025, Nijmegen, Netherlands

ACM ISBN 979-8-4007-1567-9/2025/06
https://doi.org/10.1145/3724363.3729109

1 Introduction

Generative AI models, like large language models (LLMs) and text-to-image (TTI) models, have a breadth of uses. Although many users' interactions with these models happen in the context of a direct chat interface (*e.g.,* ChatGPT), these models can also be used to power intelligent *components* of software applications, especially when used alongside prompt templates. Prompt templates (as seen in novel UIs that incorporate features powered by generative AI, *e.g.,* [17], [10], [22]) contain instructions for a particular task, interspersed with locations where data available during a program's use can be inserted so that an AI model can respond to user behavior.

Such interfaces combine traditional methods of software engineering with calls to prompt-based AI models. This component-based approach toward generative AI application features, compared to straightforward chatbot implementations, can enable developers to build features that are more powerful (as in [19], which orchestrates AI behaviors using a framework external to the AI) and more contextually-appropriate (as in [7], which guides an LLM to produce actionable outputs within design software). Such architectures are additionally important because they enable software designers to place *guardrails* that preclude undesired behaviors that could arise in open-ended applications (giving away an assignment solution [14] or obviating a puzzle [9]).

It is important to expose students to this new way of constructing interactive software, enabling them to explore capabilities and limitations to become thoughtful designers of future AI-powered software. Constructing, testing, and integrating prompt templates into software presents technical and logistical difficulties to students, however. We integrated AI-powered applications into a project-based UI design and development course, and built a software platform, REAGENT, to support the following goals:

- Support collaborative, iterative authorship and testing of prompt templates to be used in software prototypes
- Facilitate transition from exploration to integration, minimizing technical complexity
- Provide visibility of classwork to instructors
- Reduce administrative hurdles for course-scale deployment

We deployed REAGENT in our course and analyze students' work, revealing both opportunities and risks in engaging students in building AI-powered software.

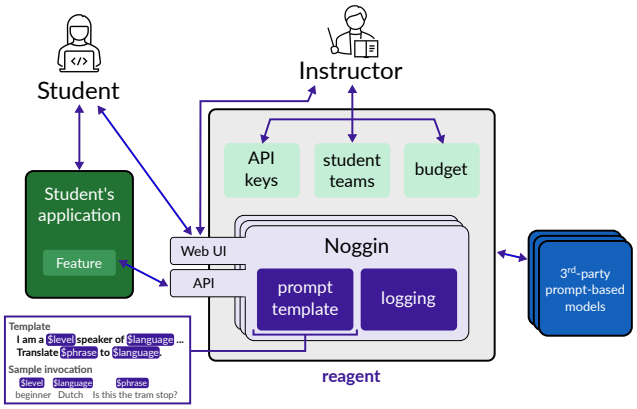


Figure 1: Noggins created within REAGENT can be configured by students, then used within application code to make requests to third-party promptable AI models.

2 Related Work

CS educators are currently grappling with impacts of AI on curriculum in a number of ways. There is much recent work exploring students' experiences using commercial generative AI products, such as ChatGPT and GitHub Copilot [5]. For example, students use LLMs to assist with code debugging [13] or in authoring code or assignment solutions [11, 13]. The research community has also investigated the development of custom tooling designed to help teach computer science material in pedagogically effective ways, or to scale teaching resources [14, 23].

In contrast, other work has focused on teaching students how to introduce intelligent features to their *own* software through traditional machine learning approaches (training, testing, and deploying a model) [6, 12]. Prompt-based models provide for an even lower barrier to entry for software integration, offering description-based invocation of a wide range of model capabilities [4, 15, 18].

Our perspective focuses on helping students understand how to integrate generative AI functionality into open-ended software design projects. This motivation connects to other research that aims to use generative AI as a design material (*e.g.,* [20, 21]), allowing designers to prototype UI behavior with little technical skill. We focus on CS students with a design task that requires implementing functional software that mediates both intelligent UI features and traditional user interactions through code.

3 Course Context

We teach a User Interface Design and Development course targeted at juniors and seniors within the CS program at a US-based research university. In the course, students learn about principles of HCI and UI design, then put those principles into practice in homework assignments and a final open-ended team project.

4 REAGENT

We identified an opportunity to facilitate students' exploration of AI through the development of a course technology platform. REAGENT is a hosted, open source Web tool[1] that offers students a rich, collaborative authoring interface for prompt-based AI models, then automatically stands up a "batteries-included" API gateway that can be used from any code.

4.1 Design goals

Our goals in designing REAGENT were inspired by our prior experience as instructors, both of this course and of other design and programming courses.

4.1.1 Support collaborative, iterative authorship and testing. Directing LLMs and TTI models to perform particular tasks is possible through just natural language. We wanted to create an environment for students to experiment with authoring prompts and evaluating model output in advance of writing code. The eventual goal of integrating prompts into software requires a skillset beyond writing one-off prompts, however: students need to author reusable prompt *templates* that operate on diverse inputs at program runtime. Thus, we designed REAGENT to empower students to test their prompts on varying inputs, refine their templates iteratively, and engineer prompts collaboratively with peers and instructors.

4.1.2 Facilitate transition from exploration to integration. As instructors, we have observed that even simple (non-AI) APIs typically require some technical skill to set up and use, requiring that students configure and manage API keys, understand documentation, and format inputs and outputs to "glue" the API into student code. These are valuable skills to teach CS students but are often incidental to an assignment's learning objectives and can consume an inordinate amount of students' debugging time. We sought to build an approachable ecosystem that could support courses targeted even at novice programmers, *e.g.,* hardware design and prototyping courses in which students typically write only "glue" code.

Prompt templates created within the REAGENT UI, therefore, are immediately available in a simple-to-use API whose invocations are visible in the same place and form as manual runs. To overcome information barriers (as in [16]), students need to feel empowered to inspect their programs. When something goes wrong in an intelligent software feature, the application boundary between the student's code and the AI model is a valuable starting point for debugging: did the correct data make it into the prompt from the software? Did the AI respond sensibly, and in a format the student's code can use? We designed REAGENT to provide visibility into constructed prompt inputs and AI outputs for all API calls, supporting students in debugging their code.

Prompt-based AI models, given their flexibility and ease of use, are a unique target for an "out of order" approach; we believe that students are ready to experiment with including intelligent features in their applications even before learning to interface with an API. Additionally, even when large AI models are not strictly necessary to carry out a task (*e.g.,* sentiment analysis), prompt-based models can enable fast, low-commitment prototyping with AI features.

4.1.3 Provide visibility to instructors. Because prompts are located in REAGENT rather than in student code, it is important for instructors to be able to see and potentially make edits to prompts, *e.g.,* when grading or when demonstrating ideas. Introducing a software architecture boundary at the prompt template level enables instructors to inspect prompts in isolation, supporting instruction related

[1] https://acelab.berkeley.edu/projects/reagent

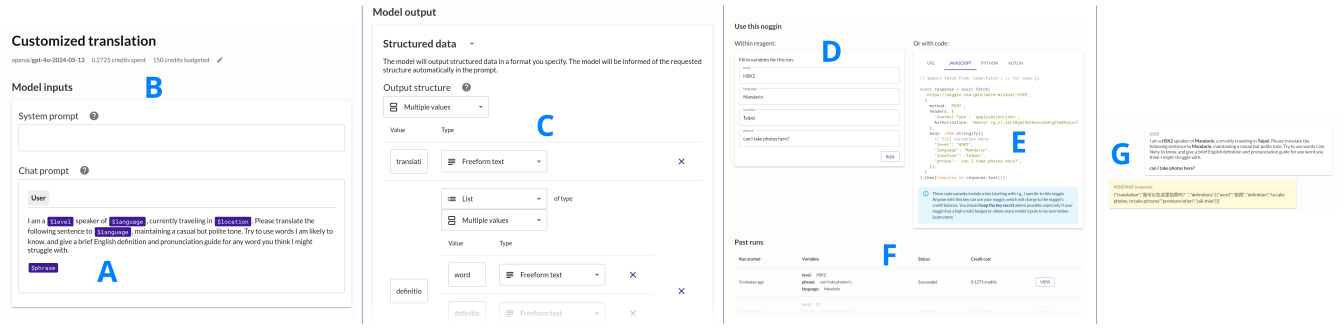


Figure 2: Screenshots of various components of REAGENT's UI. A: rich prompt editing interface; B: budget management; C: output structure editor; D: a form with a field for each variable; E: a code snippet that invokes the noggin; F: run history of this noggin; G: a run visualized in "chat" style.

to prompt engineering, which has its own pedagogical considerations separate from student-authored code. Instructors can also leverage usage history, rather than just students' final prompts, to better understand how students are forming conceptual models of generative AI, which we demonstrate in Sections 6 and 7.

4.1.4 Reduce administrative hurdles. Running LLMs and TTI models requires large amounts of compute power, typically offloaded using an API to a cloud machine. We centralize billing for all course-related use so that we could pay for students' experimentation without each student needing to configure payment details, which would pose a logistical barrier as well as a financial one. With REAGENT, instructors can manage budgets both for individual students and for teams. Additionally, building a backend-agnostic system enables rapid switching between models, reducing the friction of experimentation. With centralized billing, students can use any model regardless of its backend and how it is paid for.

4.2 Noggins: Creating Single-Purpose Tools within REAGENT

4.2.1 Authoring APIs. After logging in with a GitHub account, students are able to create a new *noggin*, a single-purpose tool backed by a chosen AI model (from the supported backends OpenAI [2], Anthropic [3], and Replicate [1]), alongside a stored prompt template and model configuration. REAGENT supports both text and image output.

Noggins' prompt templates are authored using a rich prompt editor that supports chat turns (for use with "chat"-style LLMs) and typed template variable placeholders (Figure 2, A). When authoring a prompt template, students implicitly construct an API input interface through the insertion of text or image variables into their prompt. For models that support output structure specification, REAGENT provides a visual editor to customize the output format of the noggin (Figure 2, C).

4.2.2 Using your Noggin. Students can experiment with their noggin's API in REAGENT's "Use" pane (Figure 2, D). Students can specify each noggin's budget (specified in US cents) individually, subject to a global instructor-chosen maximum over all of their budgets; runs that would go over-budget are canceled. Responses are streamed to a view that also shows exactly the prompt as it was sent to the model (Figure 2, G). Students are provided with pre-filled code snippets that can be used to invoke the noggin, either through a direct URL or a POST request in code (Figure 2, E). The noggin's API inputs match the variables from the prompt editor, effectively creating an API that abstracts away the underlying AI model and instead presents a single-purpose tool.

Some API features were designed to mitigate risks from API keys being leaked by students, *e.g.,* to public GitHub repositories: each noggin's API key can be used only for that particular noggin, inputs are truncated to prevent overly costly request parameters, and noggin budgets are capped by default so that even leaked keys cannot overspend. We did experience a student reporting the accidental leakage of a REAGENT API key on a public code repository, and we reassured the student that no harm was done.

The API backing each noggin is designed to be robust, searching for API keys and variable inputs wherever they may be present in the request (in URL parameters as well as in URL- or JSON-encoded POST data) and accepting images in various formats (data URLs, inferring image types through their payload data when the MIME prefix is missing, or external URLs fetched by the API server). Features such as this added API robustness and the API's pre-emptive run cost prediction are designed to offer guardrails and assistance, especially for students without much experience using APIs.

REAGENT shows students every run of the noggin, including those made through the API. This provides a default logging baseline that can help debug and attribute errors, showing exactly what was sent to and received from the AI.

4.2.3 Collaboration. Students are added by their instructors to an "organization" within REAGENT, which centralizes API billing by routing noggin requests through an instructor-specified API key. Course staff can choose how much total budget should be permitted across all of a member's noggins.

Instructors can form student *teams* with their own budget limits separate from individual budget limits. Team noggins are shared and editable by all team members. REAGENT's rich prompt editor supports live collaboration (in the style of Google Docs), allowing for multiple students or instructors to edit prompts concurrently during the design and debugging processes.

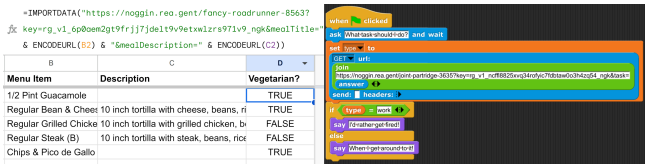


Figure 3: Left: A noggin is invoked from within Google Sheets. Right: A different noggin is invoked within a Snap! program, with its output used to alter control flow.

4.2.4 Interoperability. Because REAGENT exposes each noggin as a simple HTTP API, noggins can be used portably even in environments where it is inconvenient to craft complex HTTP requests. For example, Figure 3 demonstrates noggins being invoked within Google Sheets and the block-based programming language Snap! [8] with just an HTTP GET request, using REAGENT to "program" the behavior of the AI backend. Since the API is configured interactively within the REAGENT UI, the actual invocation of the noggin is simple enough for almost any programming-enabled environment.

5 Programming Assignment

We developed an assignment to introduce students both to REAGENT and to using AI models within code. In four tasks, students customize the behavior of AI models to fit into existing software, using AI to create text and image processing tools that improve UIs. The assignment serves as a sandbox where students can experiment and begin to learn the considerations, potential, and shortcomings of using AI components within software. Most students found this to be an easy assignment; although there is iterative work in prompting the AI to produce useful responses, this is forgiving compared to typical programming (there are no syntax errors in a prompt!).

Students were provided with starter UI code for each task, each containing basic functionality but no intelligent features. Students implemented intelligent features into UIs including text summarization, a recommendation system, an image generator, and a tool that extracts form data from a photograph *(This programming assignment is available as supplementary material*[2]*).*

The assignment introduces the flexibility of pre-trained models (*e.g.,* the model's ability to understand non-English text despite being prompted in English), the use of model output beyond displaying it to users (*e.g.,* to take UI actions), the use of visual inputs and outputs, and the potential for task failure and UI design implications. The assignment steers away from uses of AI that may have difficult-to-notice flaws (*e.g.,* tasks especially prone to hallucination); we use more focused instruction during class time to address limitations and ethical issues surrounding AI use. Students write a reflection on UI design needs specific to AI and on potential downsides of building intelligent features in this way. After completing the assignment, students begin to design and integrate AI features into their final team projects. In these open-ended projects, students implement working prototypes of software they have designed to target the discovered needs of a chosen user group.

[2]https://acelab.berkeley.edu/projects/speedy-smarts/

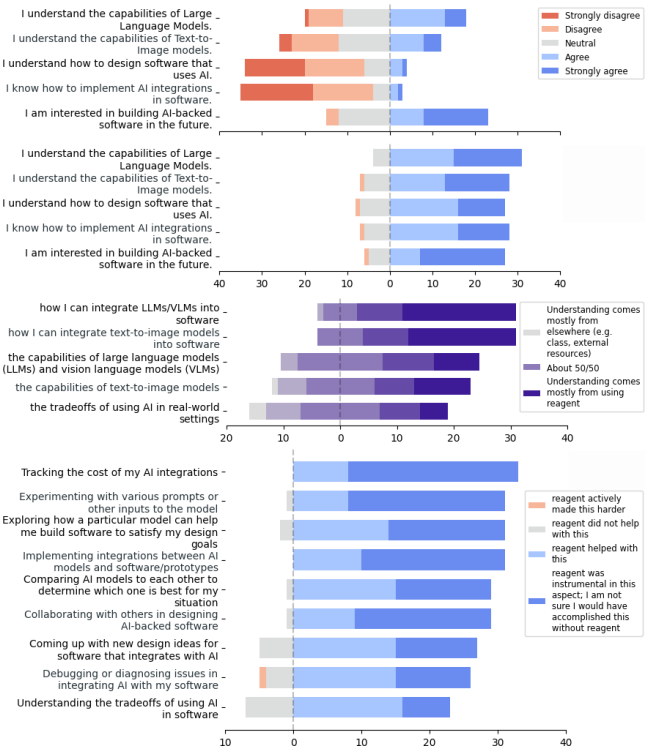


Figure 4: (a): Survey responses from before students begin their first AI-related assignment and exposure to REAGENT. (b): The same questions, asked at the end of the course. (c): Post-survey of student perception of how much their understanding comes from REAGENT use. (d): Post-survey of student perception of how much REAGENT assisted in tasks related to the integration of AI in software.

6 Observations

After a successful pilot deployment in spring 2024, we conducted an IRB-approved research study in our summer 2024 semester, collecting data about REAGENT use. All students were permitted to use REAGENT for their work, and course staff members were not informed of the identities of consenting students until after final grades were submitted. Of 69 students who completed the course, 39 consented to the use of their responses and work in this research. We surveyed students before their initial introduction to REAGENT and at the end of the course; 38 participants responded to this initial survey, and 35 of those additionally responded to the final survey. We also collected usage history from REAGENT's database detailing participants' use of REAGENT in their coursework.

6.1 Building and Exploring with REAGENT

Students made heavy use of REAGENT, both in individual homework and their open-ended final project. Students reported an increased understanding of how to integrate large AI models into their work. Before the introduction of REAGENT, only 3 of 38 respondents agreed or strongly agreed with the statement "I know how to implement AI integrations in software"; at the end of the semester, this grew to 28 of 35 respondents. Because we provided in-class instruction on AI

in addition to having students experiment using REAGENT, we also asked students at the end of the semester where their understanding of various topics came from (see Figure 4c); these responses skewed heavily toward REAGENT offering significant benefits. REAGENT assisted with many tasks in the integration of AI into software, shown in the responses in Figure 4d (participants who selected "N/A" are excluded from each bar).

Students were generally very positive in their evaluation of REAGENT and its impact on their process. Some benefit of using REAGENT came from its convenience for experimenting with large AI models' capabilities before integrating into code; 31 participants reported that REAGENT helped with or was instrumental in experimenting with prompts (Figure 4d, row 2).

Thirty-one students reported that REAGENT helped implement their AI integrations (Figure 4d, row 4): *"What surprised me most is how easy it was! The fact that Noggin gives you JS to copy-paste into your code is very convenient, and it kind of feels accessible and like anyone can do it. I was kind of intimidated to use AI for our final team project, but this helped me get more comfortable with it."* - **P2**

Comparing models using REAGENT was helpful to 29 students (Figure 4d, row 5), and students appreciated that they could quickly switch between AI models, regardless of backend: *"I think reagent is very useful in that I can test different models and compare them based on needs, credit cost, and model proficiency. It is very useful and easy to use, and honestly broke down the barrier/fear of implementing AI as it made tasks so much easier and simpler."* - **P38**

Configuring prompts separately from the invoking code was also considered valuable, and REAGENT provided an easy on-ramp into using the noggin in code: *"I found the ability to use it in the browser really helpful."* - **P20**; *"I can easily prompt the model, and the code for using the model as backend is already provided."* - **P9**

Students also helped to validate certain design decisions as helping them achieve their goals and build understanding, as described in Figure 4d and in participant comments: *"limiting the amount of credits assigned to a particular model can help mitigate the potential losses from leaked api keys."* - **P41**; *"Collaborate within a team is more efficient with reagent."* - **P33**; *"I liked that I was able to track and compare the costs of an AI integration because I hadn't considered how cost was also a real-world tradeoff. Instead of spamming the calls to the model, I tried to be smart about conserving my resources and careful in tuning up my prompts."* - **P39**

6.2 Student experiences with AI

Through analysis of REAGENT use and student projects, we observed how students design and develop with AI-backed components.

6.2.1 Leveraging model flexibility. Students frequently authored noggins in ways that, intentionally or otherwise, took advantage of the flexibility of large, pre-trained AI models.

When designing an LLM-backed API, the *interface* of the API can be chosen by the developer, with the output structure expressed to the model in a number of ways. This can flex to however the student finds it natural to instruct the LLM or write their consuming code. In an optional homework task attempted by 22 participants, students needed an LLM to output both a discrete choice and a "reasoning" for this choice, parsing the response to separate these two outputs. Students approached this in different ways: six students used REAGENT's "structured JSON output" feature; five described a JSON-parsable output format manually in the prompt; nine prompted the model to use some other format (*e.g.,* separating outputs with a colon or newline); and two used one noggin to make the choice, then passed the result into a second "explainer" noggin. In addition, four of these students offered some examples matching the requested output format. Although REAGENT assists students in scaffolding custom APIs, it does not over-prescribe the chosen API interfaces, instead permitting the choice afforded by large AI models' flexibility in how the results of a model call will be used.

Model robustness can also help overcome inconsistencies in API invocation, such as inputs in an unexpected format. In students' final projects, we observed cases where the format of a noggin variable's value was slightly different between use in the REAGENT UI and later use in code; for example, one student used the text `12:53 P.M. 8/6/24` in a prompt variable when testing in the REAGENT UI and subsequently sent the text `8/6/2024, 11:25:52 PM` to the noggin's API, presumably generated by their code. The underlying LLM responded in the same way both times despite the differing date formats between requests.

Some prompt templates included minor errors, such as typos, missing or extra whitespace, or grammatical errors in instructions. We found that these errors largely did not prevent the model from carrying out requested tasks, demonstrating a robustness not generally present when invoking non-AI-backed systems.

6.2.2 Errors in Invocation. Most participants (35 of 39), at some point in their individual work, made at least one request to a noggin that had some missing or malformed variable. Invocations with missing variable values often caused the prompt to be nonsensical. Some may have been intentional (*e.g.,* from students testing how their UI behaves when the user's input is empty), but other nonsense invocations were clearly mistakes. For example, 7 participants sent the literal string `[object Object]` — a common mistakenly-generated string in JavaScript code — to a noggin. Logs of whether the noggin was invoked using the REAGENT UI or the hosted API (implying an invocation from student code) showed that malformed inputs frequently appear right at the moment when requests switch from being made in the UI to being made in the code, further suggesting that these are integration errors. Critically, such invocations often *did* pass through to the underlying AI model, since they did not contain errors in making the request to the noggin's API.

7 Discussion

Many of our observations of student work and of REAGENT use offer lessons for instructors who wish to incorporate AI-backed prototyping into their courses.

7.1 Approachable Creation of Software Components

REAGENT's noggins suggest an architectural boundary that is especially valuable for students new to developing with AI models or those less skilled at programming. By isolating and experimenting with a prompt template separately from a codebase, students can more easily observe what a model is doing and its role in the software's architecture. The exploration afforded by REAGENT allows

students to build an understanding of costs of using AI models, as well as capability limitations such as risks of hallucination.

It was common for students to use natural language techniques to prompt an LLM to output in a particular format, or even to use language instructions rather than programmatic techniques to achieve certain behavior, effectively "programming in the prompt".

For example, one participant directed an LLM's output using this instruction: `Also, remember to change all "/n" in the text into "<br>"! This is also IMPORTANT!! Because we are using the text in HTML.` Notably, this prompt contains a typo (using `/n` rather than `\n` to refer to the escaped newline in the JSON output). If this exact instruction had been expressed instead with code (*e.g.,* with a string `.replace()` call), it would not have worked as the student intended, but the LLM's outputs when the prompt included this snippet were in line with the student's intent.

Another participant used a JSON object obtained from the Yelp API as an input variable to a noggin, authoring the following instruction to the model: `Please filter the restaurants that are currently open`. The array could instead have been filtered programmatically, using the Boolean field called `is_open_now`.

These code-like instructions within prompts suggest a fuzzy boundary in hybrid code/prompt applications, in which students can either use code or a runtime AI call to solve a task. Instructors should consider how to teach students about this choice, given contextual considerations such as students' programming ability or tradeoffs between robustness, efficiency, and prototyping speed.

7.2 Robustness is a Risk

As discussed in Section 6.2.1, large AI models offer a robustness that can help applications deal with a wide range of inputs. A traditional API may slow down a user who doesn't understand the nature of the inconsistency and simply receives an error, but a noggin will accept imperfect inputs. However, we observed that this robustness presents a risk: rather than responding with an error message when a mistake is made, a model may simply "do its best", resulting in suboptimal or even completely fabricated results.

Not every student caught invocation errors before finishing their work. For example, one participant left an invocation error in their final submission of the recommendation homework task (sending `[object Object]`), and the noggin continued to dutifully respond with an arbitrary "recommendation"! In another team's final project demonstration, a "document summarization" feature tended to write vague summaries, and the student simply reported that the feature "doesn't always work so well" — upon our later inspection of the API call, the input had not been populated at all, and the summary had been a hallucination.

In cases where errors were ultimately corrected, we cannot know for sure whether REAGENT's input visibility features assisted students in locating the bug, except when reported: *"I find it most helpful when model output isn't quite working, I can check whether something is wrong with the input using reagent"* - **P41**. However, we believe that REAGENT helped many students overcome a baseline difficulty in noticing and diagnosing these issues.

7.3 Building Deeper Understanding

Enabling students to experiment directly with AI models permits them to form nuanced understanding of AI capabilities. As **P39** describes: *"I learned in class that LLMs are not good at doing math, so I just thought it couldn't compute equations. After fiddling around with Reagent, turns out that LLMs also cannot count . . . I didn't realize counting was part of the math it couldn't do. I've been able to get a better understanding of concepts in class by actively using reagent."*

Not all conceptual misunderstandings are easy for students to self-diagnose, however. While providing API visibility can help students uncover *invocation errors*, robustness presents an additional risk in the formation of students' *conceptual models* of the use of AI models. For example, we observed students prompting non-tool-use LLMs to look up information on the Internet; students incorrectly prompting Stable Diffusion using direct instructions (*e.g., "Generate an image that..."*); and students incorrectly formatting chat-based prompts, *e.g.,* placing user queries in "Assistant:" blocks.

Even in these cases, models often respond similarly to how they would when prompted correctly. Therefore, these errors can go unnoticed by students, more easily than invocation errors that can be noticed by inspecting model inputs. We often observed models responding to this kind of query with hallucinated information (in the case of demands for Internet lookups) or subpar results (in the case of incorrect TTI prompt structure). Students sought less aid from course staff in integrating their code with REAGENT than the authors have come to expect from courses that make use of traditional APIs. Where a mistake in a traditional API typically results in an error that causes students to seek help in office hours, AI APIs may not complain at all.

That does not preclude errors, however. Since model robustness can obscure fundamental misunderstandings, instructor visibility into AI prompts is invaluable for keeping an eye on students' mastery without needing to dig into student code. One potential direction of future work would be to give educators high-level visibility into certain types of suspect prompting strategies, like "linters" for the fuzzier world of AI prompting, to provide more proactive insights into where misunderstandings need to be corrected.

Other features of REAGENT designed primarily to address logistical challenges proved to be valuable in building targeted understanding. Routing requests through REAGENT helps students switch quickly between AI models, helping to focus attention on more general principles of AI capabilities, rather than on specifics about provider APIs. Clear, centralized budgets help students to find the right tools for open-ended projects without needing to shuffle budget between platforms and to reason about AI model costs within the context of their projects.

8 Conclusion

We introduce REAGENT, a tool that enables students to explore integrating AI into software prototypes. Our deployment provided value to students, reducing friction and supporting collaborative engineering of intelligent applications. REAGENT offered insight into students' use and understanding of AI models, and we hope to keep using REAGENT as the AI landscape continues to evolve.

Acknowledgments

This work was supported in part by grant #00005919 from the California Education Learning Lab (calearninglab.org), an initiative of the California Governor's Office of Planning and Research. We also thank Shm Garanganao Almeda for assisting with our study.

References

[1] 2025. *Docs - Replicate*. https://replicate.com/docs/

[2] 2025. *Overview - OpenAI API*. https://platform.openai.com

[3] 2025. *Welcome to Claude*. https://docs.anthropic.com/en/docs/welcome

[4] Stephen H. Bach, Victor Sanh, Zheng-Xin Yong, Albert Webson, Colin Raffel, Nihal V. Nayak, Abheesht Sharma, Taewoon Kim, M. Saiful Bari, Thibault Fevry, Zaid Alyafeai, Manan Dey, Andrea Santilli, Zhiqing Sun, Srulik Ben-David, Canwen Xu, Gunjan Chhablani, Han Wang, Jason Alan Fries, Maged S. Al-shaibani, Shanya Sharma, Urmish Thakker, Khalid Almubarak, Xiangru Tang, Dragomir Radev, Mike Tian-Jian Jiang, and Alexander M. Rush. 2022. PromptSource: An Integrated Development Environment and Repository for Natural Language Prompts. https://doi.org/10.48550/arXiv.2202.01279 arXiv:2202.01279 [cs].

[5] Brett A. Becker, Paul Denny, James Finnie-Ansley, Andrew Luxton-Reilly, James Prather, and Eddie Antonio Santos. 2023. Programming Is Hard - Or at Least It Used to Be: Educational Opportunities and Challenges of AI Code Generation. In *Proceedings of the 54th ACM Technical Symposium on Computer Science Education V. 1 (SIGCSE 2023)*. Association for Computing Machinery, New York, NY, USA, 500–506. https://doi.org/10.1145/3545945.3569759

[6] Jie Chao, Bill Finzer, Carolyn P. Rosé, Shiyan Jiang, Michael Yoder, James Fiacco, Chas Murray, Cansu Tatar, and Kenia Wiedemann. 2022. StoryQ: A Web-Based Machine Learning and Text Mining Tool for K-12 Students. In *Proceedings of the 53rd ACM Technical Symposium on Computer Science Education V. 2 (SIGCSE 2022)*. Association for Computing Machinery, New York, NY, USA, 1178. https://doi.org/10.1145/3478432.3499267

[7] Peitong Duan, Jeremy Warner, Yang Li, and Bjoern Hartmann. 2024. Generating Automatic Feedback on UI Mockups with Large Language Models. In *Proceedings of the 2024 CHI Conference on Human Factors in Computing Systems (CHI '24)*. Association for Computing Machinery, New York, NY, USA, 1–20. https://doi.org/10.1145/3613904.3642782

[8] Brian Harvey, Daniel D. Garcia, Tiffany Barnes, Nathaniel Titterton, Daniel Armendariz, Luke Segars, Eugene Lemon, Sean Morris, and Josh Paley. 2013. SNAP! (build your own blocks) (abstract only). In *Proceeding of the 44th ACM technical symposium on Computer science education (SIGCSE '13)*. Association for Computing Machinery, New York, NY, USA, 759. https://doi.org/10.1145/2445196.2445507

[9] Nicholas Jennings, Han Wang, Isabel Li, James Smith, and Bjoern Hartmann. 2024. What's the Game, then? Opportunities and Challenges for Runtime Behavior Generation. In *Proceedings of the 37th Annual ACM Symposium on User Interface Software and Technology (UIST '24)*. Association for Computing Machinery, New York, NY, USA, 1–13. https://doi.org/10.1145/3654777.3676358

[10] Peiling Jiang, Jude Rayan, Steven P. Dow, and Haijun Xia. 2023. Graphologue: Exploring large language model responses with interactive diagrams. In *Proceedings of the 36th annual ACM symposium on user interface software and technology (Uist '23)*. Association for Computing Machinery, New York, NY, USA. https://doi.org/10.1145/3586183.3606737 Number of pages: 20 Place: San Francisco, CA, USA tex.articleno: 3.

[11] Ishika Joshi, Ritvik Budhiraja, Harshal Dev, Jahnvi Kadia, Mohammad Osama Ataullah, Sayan Mitra, Harshal D. Akolekar, and Dhruv Kumar. 2024. ChatGPT in the Classroom: An Analysis of Its Strengths and Weaknesses for Solving Undergraduate Computer Science Questions. In *Proceedings of the 55th ACM Technical Symposium on Computer Science Education V. 1 (SIGCSE 2024)*. Association for Computing Machinery, New York, NY, USA, 625–631. https://doi.org/10.1145/3626252.3630803

[12] Gloria Ashiya Katuka, Srijita Chakraburty, Hyejeong Lee, Sunny Dhama, Toni Earle-Randell, Mehmet Celepkolu, Kristy Elizabeth Boyer, Krista Glazewski, Cindy Hmelo-Silver, and Tom Mcklin. 2024. Integrating Natural Language Processing in Middle School Science Classrooms: An Experience Report. In *Proceedings of the 55th ACM Technical Symposium on Computer Science Education V. 1 (SIGCSE 2024)*. Association for Computing Machinery, New York, NY, USA, 639–645. https://doi.org/10.1145/3626252.3630881

[13] Majeed Kazemitabaar, Xinying Hou, Austin Henley, Barbara Jane Ericson, David Weintrop, and Tovi Grossman. 2024. How Novices Use LLM-based Code Generators to Solve CS1 Coding Tasks in a Self-Paced Learning Environment. In *Proceedings of the 23rd Koli Calling International Conference on Computing Education Research (Koli Calling '23)*. Association for Computing Machinery, New York, NY, USA, 1–12. https://doi.org/10.1145/3631802.3631806

[14] Majeed Kazemitabaar, Runlong Ye, Xiaoning Wang, Austin Zachary Henley, Paul Denny, Michelle Craig, and Tovi Grossman. 2024. CodeAid: Evaluating a Classroom Deployment of an LLM-based Programming Assistant that Balances Student and Educator Needs. In *Proceedings of the 2024 CHI Conference on Human Factors in Computing Systems (CHI '24)*. Association for Computing Machinery, New York, NY, USA, 1–20. https://doi.org/10.1145/3613904.3642773

[15] Omar Khattab, Arnav Singhvi, Paridhi Maheshwari, Zhiyuan Zhang, Keshav Santhanam, Sri Vardhamanan, Saiful Haq, Ashutosh Sharma, Thomas T. Joshi, Hanna Moazam, Heather Miller, Matei Zaharia, and Christopher Potts. 2023. DSPy: Compiling Declarative Language Model Calls into Self-Improving Pipelines. https://doi.org/10.48550/arXiv.2310.03714 arXiv:2310.03714 [cs].

[16] Amy J. Ko, Brad A. Myers, and Htet Htet Aung. 2004. Six learning barriers in end-user programming systems. In *2004 IEEE symposium on visual languages - human centric computing*. 199–206. https://doi.org/10.1109/VLHCC.2004.47

[17] Michelle S. Lam, Janice Teoh, James A. Landay, Jeffrey Heer, and Michael S. Bernstein. 2024. Concept induction: Analyzing unstructured text with high-level concepts using LLooM. In *Proceedings of the CHI conference on human factors in computing systems (Chi '24)*. Association for Computing Machinery, New York, NY, USA. https://doi.org/10.1145/3613904.3642830 Number of pages: 28 Place: Honolulu, HI, USA tex.articleno: 766.

[18] Krista Opsahl-Ong, Michael J. Ryan, Josh Purtell, David Broman, Christopher Potts, Matei Zaharia, and Omar Khattab. 2024. Optimizing Instructions and Demonstrations for Multi-Stage Language Model Programs. https://doi.org/10.48550/arXiv.2406.11695 arXiv:2406.11695 [cs].

[19] Joon Sung Park, Joseph O'Brien, Carrie Jun Cai, Meredith Ringel Morris, Percy Liang, and Michael S. Bernstein. 2023. Generative Agents: Interactive Simulacra of Human Behavior. In *Proceedings of the 36th Annual ACM Symposium on User Interface Software and Technology (UIST '23)*. Association for Computing Machinery, New York, NY, USA, 1–22. https://doi.org/10.1145/3586183.3606763

[20] Savvas Petridis, Michael Xieyang Liu, Alexander J. Fiannaca, Vivian Tsai, Michael Terry, and Carrie J. Cai. 2024. In Situ AI Prototyping: Infusing Multimodal Prompts into Mobile Settings with MobileMaker. https://doi.org/10.48550/arXiv.2405.03806 arXiv:2405.03806 [cs].

[21] Savvas Petridis, Michael Terry, and Carrie J Cai. 2024. PromptInfuser: How tightly coupling AI and UI design impacts designers' workflows. In *Proceedings of the 2024 ACM designing interactive systems conference (Dis '24)*. Association for Computing Machinery, New York, NY, USA, 743–756. https://doi.org/10.1145/3643834.3661613 Number of pages: 14 Place: IT University of Copenhagen, Denmark.

[22] Sangho Suh, Bryan Min, Srishti Palani, and Haijun Xia. 2023. Sensecape: Enabling multilevel exploration and sensemaking with large language models. In *Proceedings of the 36th annual ACM symposium on user interface software and technology (Uist '23)*. Association for Computing Machinery, New York, NY, USA. https://doi.org/10.1145/3586183.3606756 Number of pages: 18 Place: San Francisco, CA, USA tex.articleno: 1.

[23] J. D. Zamfirescu-Pereira, Laryn Qi, Björn Hartmann, John DeNero, and Narges Norouzi. 2024. 61A Bot Report: AI Assistants in CS1 Save Students Homework Time and Reduce Demands on Staff. (Now What?). https://doi.org/10.1145/3641554.3701864 arXiv:2406.05600 [cs].