OpenTutor: Designing a Rapid-Authored Tutor that Learns as you Grade

Benjamin D. Nye[1], Rushit Sanghrajka[2], Vinit Bodhwani[1], Martin Acob[1], Daniel Budziwojski[1], Kayla Carr[1], Larry Kirschner[1], William R. Swartout[1]
[1]Institute for Creative Technologies, University of Southern California
[2]School of Computing, University of Utah
nye@ict.usc.edu

Abstract

Despite strong evidence that dialog-based intelligent tutoring systems (ITS) can increase learning gains, few courses include these tutors. In this research, we posit that existing dialog-based tutoring systems are not widely used because they are too complex and unfamiliar for a typical teacher to adapt or augment. OpenTutor is an open-source research project intended to scale up dialog-based tutoring by enabling ordinary teachers to rapidly author and improve dialog-based ITS, where authoring is presented through familiar tasks such as assessment item creation and grading. Formative usability results from a set of five non-CS educators are presented, which indicate that the OpenTutor system was relatively easy to use but that teachers would closely consider the cost benefit for time vs. student outcomes. Specifically, while OpenTutor grading was faster than expected, teachers reported that they would only spend any additional time (compared to a multiple choice) if the content required deeper learning. To decrease time to train answer classifiers, OpenTutor is investigating ways to reduce cold-start problems for tutoring dialogs.

Dialog-based intelligent tutoring systems (ITS) have shown strong learning gains across a variety of domains, on the order of 0.4σ to 1.5σ (Kulik and Fletcher 2016; Nye, Graesser, and Hu 2014). Unfortunately, despite over a decade of evidence of effective dialog-based ITS, relatively few content modules include these tutors. The primary bottleneck for dialog-based ITS has been content authoring: typically computer-science experts or even AI experts must be tightly involved with creating new tutoring activities.

In this research, we posit that existing dialog-based tutoring systems are not widely used because creating or modifying content for them is too complex and unfamiliar for a typical teacher. Specifically, in Nye et al. (2014) we noted that most authoring tools include functionality to author multiple different areas, such as the domain content, the pedagogy and dialog strategies, and the user interface (UI). Based on experience collaborating with non-technical authors on tutoring dialogs, our intuition was that such authors should be able to create tutoring dialogs if they only need to author and edit *domain content* rather than UI or dialog models. Unfortunately, natural language-based intelligent tutors are often initially inaccurate in their assessment of student answers and require expert computing skills to optimize features or improve accuracy. To address this issue, we investigate an approach where teachers improve the quality of dialogs through a familiar task: grading students' answers.

Background and Related Work

This project draws from two different areas of research and development: ITS authoring tools and broader non-technical authoring tools for interactive dialogs. ITS research on authoring tools for tutoring dialogs remains relatively uncommon, representing under 10% of tools in a recent systematic review (Dermeval et al. 2018).

At least one effort, AutoTutor Lite, has been modestly successful in that external research institutions have successfully developed their own tutoring content (Wolfe et al. 2013) and it has been integrated into the GIFT open source framework (Wang et al. 2020). AutoTutor Lite primarily enables authoring dialog content (e.g., questions, responses, feedback), and also configures technical parameters (e.g., feedback thresholds) and UI configuration such as associated media or multiple tutoring agents. It also offers the ability to develop full AutoTutor scripts (e.g., with production rules) as an advanced feature. A related effort, the Rapid Form-Based Authoring Tool for AutoTutor attempted a step-by-step walk-through to simplify this process (Nye et al. 2014), but was ultimately not successful because it was hard to revise dialog content and content revision is often more time consuming than initial authoring.

In the commercial space, non-technical authors have developed open response interactions for education and for game development. Game content tools have focused on script writers as their authors, who adapt their skills to developing text scripts with markup (e.g., Twine, Inkle, Yarnspinner) or visual graphs such as Chatmapper (Neil 2018). These primarily represent branching or looping interactions, driven by multiple choice inputs or evaluating parameters conditionally. As such, they train non-technical authors to optionally add procedural programming syntax. A common theme for these tools are that they add functionality to authoring forms they already know (e.g., scenes, screenplay scripts, graphs of cards).

In the education space, teachers do not author interactive conversations. However, they do create item sequences which sometimes have feedback and branching. Many learning management systems (LMS) implement open-response systems which allow multiple attempts and feedback based on which keywords or approximate matching. As a complementary tool, most LMS support manual grading of free text answers (e.g., Canvas Speedgrader, Blackboard rubrics), with tabular rubrics to rapidly grade text input. Overall, teachers author declarative information (e.g., content, labeling) rather than nested procedures and conditions, but these LMS tools are still relevant to aspects of dialog authoring.

OpenTutor Design

The central design goal for OpenTutor is to provide a system that teachers can quickly create and refine conversational ITS dialogs, while still retaining the well-established learning gains of expectation-misconception conversational tutoring (Graesser 2016; Nye, Graesser, and Hu 2014). OpenTutor is built to develop dialogs in a four-step cycle:

1. Authoring: Teacher authors and revises an dialog with an open-ended question and a set of expectations (concepts), which each have a set of hints.
2. Tutoring Test-Sessions: A set of tutoring sessions are completed to understand how users respond to the tutor.
3. Grading: Teachers grade these tutoring test-sessions, generating labels and which determine an official grade for that session or confirm the grade automatically generated by OpenTutor (which is visible when grading).
4. Machine Learning: The teacher pushes a Train button to use machine learning to improve the dialog classifier, improving tutoring quality and automatic grading.

By using this process, OpenTutor is distinct in three ways:

- Familiar Tasks: The OpenTutor authoring process seeks to build UI interfaces that make developing a dialog-based ITS feel like tasks that teachers are already familiar with, such as item authoring and grading.
- Online Learning: OpenTutor collects student answers as data, so that human and machine intelligence can improve the tutor based on actual student responses.
- Service-Oriented: The tutor has separate services the user interface, dialog model, and dialog classifiers. This structure is to enable open source iteration and improvements.

To maintain familiarity, OpenTutor intentionally limits its generalizability. In terms of the four-block model of ITS, authoring in OpenTutor is tightly focused on the Domain Module rather than the Pedagogy Module, Communication/UI Module, or a persistent Student Module. As compared to prior tools such as AutoTutor Lite, OpenTutor authors do not develop conditional evaluation or conversational logic. This is because, in our experience, even trained non-technical authors require substantial time to create and refine a high quality dialog flow. So then, rather than teachers authoring new conversation flows, we expect that programmers contributing to the open source code will instead implement and test dialog models. As alternate templates are developed, these could be selected by authors to use instead.

Dialog Service

The dialog service (opentutor-dialog) provides an API for tutoring dialog sessions and acts as the Pedagogy Module. The dialog service starts a tutoring session, generates system responses to user text-based answers, and reports back evaluations of the quality of the students' session performance. The user's responses are recorded for each session, to enable grading and improving the classifier.

OpenTutor's current dialog flow is adapted from a specific AutoTutor expectation-misconception template, which was used extensively in the PAL3 mobile learning coach project (Swartout et al. 2016). OpenTutor poses an open-ended question with about 1 to 6 expectations, which are concepts that are represented by an ideal answer. As the student answers, the full set of expectations are checked such that the tutor will skip any which are scored highly by the classifier. For an unmatched expectation, the tutor will provide a hint (leading question) intended to help the learner state the concept. When the student answers a hint, the tutor may either a) evaluate it as correct, give positive feedback, and move to the next expectation; b) evaluate it as matching a different uncovered expectation, give "good thought, but there's more to it" feedback, and ask another hint; or c) give neutral or negative feedback based on the evaluation and ask another hint. Since the tutor might accept answers that only loosely match the ideal answer, the tutor states the ideal answer when done giving hints for an expectation (i.e., student covered it or ran out of hints).

The tutor focuses on expectations in the order that they were authored, other than skipping expectations that have already been covered by the user's answers. By comparison, AutoTutor traditionally uses an algorithmic approach to choose the next expectation to tutor. Our experience was that instructors preferred a predictable order for tutoring, so that they could assume one concept was known before tutoring the next one. Teachers' preferred order was not based on domain knowledge, but was instead based on pedagogical domain knowledge: in one problem they might want the student to know the right answer first and then ask why, but in another problem they might want the reverse order.

For scoring and UI visualization purposes, OpenTutor returns the scores for each expectation and an overall dialog score based on their performance matching the expectations. During the dialog, OpenTutor also uses scores from two general-purpose classifiers currently based on regular expressions. These classifiers detect potential non-answers, such as metacognitive statements ("i don't know") and swearing. How the tutor responds to these general classifiers depends on the relevance of the student's input to the expectations. This means that an abusive off-topic statement ("damn you") will cause the tutor to react to profanity, but a partially correct answer which includes a swear will receive more ordinary positive or neutral feedback.

Classifier Service

The classifier service (opentutor-classifier) evaluates the quality of a student's input against a set of expectations. For every user input, the classifier outputs a classification label for relevant to every expectation (good, neutral, irrelevant) and a match score from 0 (definitely irrelevant) to 1 (definitely good) based on the classifier confidence. Unlike many tutoring dialog classifiers, OpenTutor is designed to train a dedicated classifier model for every single expectation so that each dialog can be rapidly optimized as new user answers are logged. This approach embodies the concept first explored in Nye et al. (2015), which demonstrated that for certain content a supervised-learning classifier could reach comparable accuracy to a hand-tuned classifier within under 30 labeled examples.

The current classifier is intended to provide a baseline that can be iterated and improved upon as examples from real tutoring dialogs grow over time. Based on the usefulness of support vector machines with small data (Nye, Hajeer, and Cai 2015), SVM was chosen as the baseline model. The features for each trained model depend on the data available (e.g., regex patterns will only be used if available) and can be configured, but the initial set of features are:

- Semantic Similarity (Ideal Answer): Calculates the cosine similarity of the average Word2Vec vectors for the input against the ideal answer (Mikolov et al. 2013).
- Semantic Similarity (Main Question): Calculates the similarity of the input against the main question prompt (Sultan, Salazar, and Sumner 2016). This feature can help a model to adjust for the baseline semantic similarity from answering a question with the question prompt.
- Optimal Word Alignment: A pairwise approach to semantic similarity, which calculates an average Word2Vec similarity for the best match (without replacement) of input words to the ideal answer, based on Rus et al. (2012).
- Length Ratio: The ratio of the number of input words to the number of ideal answer words (input / ideal). Initial testing indicates that this feature may reduce performance for small N due to unbalanced training sets (e.g., authors testing many short-bad answers or long-good answers), which implies the need to down-weight this feature.
- Negation Indicator: A boolean feature which indicates if an odd number of negations was detected.
- Good Regex: If provided, a set of regular expressions will be evaluated and output a score from 0 to 1 for the fraction which matched the input. This feature evaluates patterns present in good answers.
- Bad Regex: Using the same approach as Good Regex, a set of regular expressions to help detect bad or irrelevant answers can be evaluated.

The quality of the classifier is still under evaluation. Cross-validation metrics are built-in to the code base to assist with evaluating the benefits of additional labeled data. However, the quality of the current classifier appears to vary substantially depending on dialog content (e.g., accuracies ranging between 40%-100% on leave-one-out cross-validation testing). As with AutoTutor, the addition of expert-authored regular expressions substantially improves accuracy (e.g., in many cases from below 70% to over 80%). However, since authoring regex patterns contradicts the OpenTutor's core mission of "no technical skills required," this is only relevant to importing pre-existing dialogs where expressions already exist. This does imply that feature extraction which could identify these patterns could boost accuracy, which was applied in earlier work to detect ordered n-grams in earlier related work (Nye, Hajeer, and Cai 2015). However, selection of features must be balanced against usability: the training speed is currently fairly fast (e.g., < 10s) but this is likely to decrease with larger data or more computationally-expensive features.

Every dialog can store a trained machine learning model and feature set configuration for each expectation. If a dialog is not known in the classifier, the dialog information (e.g., expectation ideal answers) can be added to the classifier requests and a *default classifier* will be used. This classifier is trained on all known dialog data, which means that it is only able to estimate weights and decision thresholds based on typical similarity patterns between answers. As a result, the default classifier will be suboptimal and only appropriate for gathering initial data to train dialog-specific classifiers.

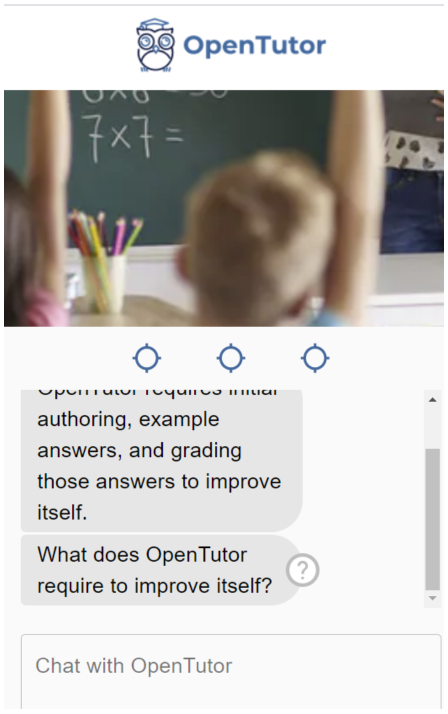
Tutoring User Interface

While the dialog model and classifier can be used as standalone services for other systems, OpenTutor also has its own built-in Tutoring UI that is immediately accessible to authors developing or sharing dialog lessons. As shown in Figure 1, the interface contains a chat message area that records the dialog between the tutor and student, a text entry frame, and a space for media (e.g., an image) above the chat. The specific dialog in the screenshots is an example used for screenshots in the step-by-step tutorial for instructors as they make their own dialog. During formative testing (described later), instructors could optionally access and execute this example, but none of them did so.

Inside the chat, each message from the tutor has an associated speech act icon (e.g., positive feedback, hint, assertion) and on the left hand side of the chat transcript. User inputs are on the right hand side. Between the media area and the chat, there are a set of circles which represent each expectation. As a user's answers match expectations, these circular icons fill to show OpenTutor's current score for each expectation. Additionally, as the tutor starts providing hints or support related to an expectation, that expectation's icon is designated with additional graphics (filling in the dot for the circle). to help learners understand the purpose of that response. At the end of the dialog, the full text of each expectation is shown in a summary panel, along with the final score level for each expectation (as shown in Figure 2).

Authoring and Grading UI

For OpenTutor to be useful, it must be easily authorable and able to improve based on user answers. To make the tool more accessible to teachers, the Authoring UI design considered comparisons from content management system tools (e.g., LMS, survey editors) and grading tools. While the Authoring UI will display on a mobile device (and screenshots shown are cropped to these dimensions), it is not optimized for editing on small mobile devices and is intended to be used with laptops or desktops.


OpenTutor

OpenTutor requires initial authoring, example answers, and grading those answers to improve itself.

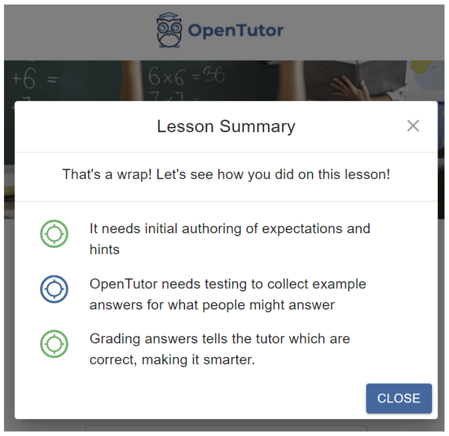
What does OpenTutor require to improve itself?

Figure 1: Default Tutoring UI

When an author logs into the system, they first see a Lesson List page where they can review the lessons they own and also view any lessons made public to them. When they create or edit a lesson, they see the Lesson Editor where they immediately see how to name the lesson, write an optional introduction statement, and author the main question. After writing the question, they add expectations. For each expectation they can author a series of hints, as shown in Figure 3. When done editing the expectations and (optionally) a closing statement, the bottom panel of the lesson has options to Save, Launch (test in the default UI), and Train (update the classifiers based on graded session answers).

After completing a lesson, authors are instructed to complete it four times, acting out different learner archetypes with the given instructions:

1. Expert Learner: Try to answer with the full answer that covers all expectations. This should be an answer that the tutor accepts fully, gives positive feedback, and ends the dialog.
2. Good Learner: Try to answer each expectation well, one at a time. If the tutor doesn't understand your good answer that the tutor should accept for that expectation, keep trying good answers to that expectation until it accepts them or it gives you the answer. Do this for each expectation.
3. Inconsistent Learner: Try to answer irregularly at varying quality, or introduce specific misconceptions that you know are a problem and students might say.
4. Struggling Learner: Try to answer incorrectly consistently, but still relevant to the overall topic. If you know any key misconceptions that students tend to say, state them here.


OpenTutor

Lesson Summary

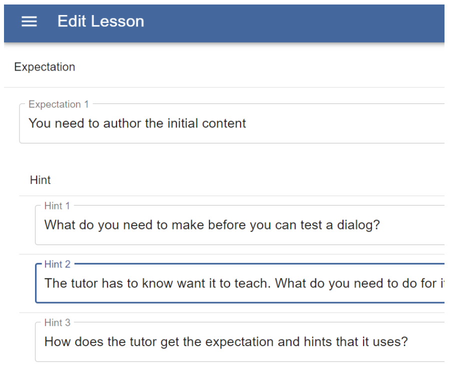
That's a wrap! Let's see how you did on this lesson!

It needs initial authoring of expectations and hints

OpenTutor needs testing to collect example answers for what people might answer

Grading answers tells the tutor which are correct, making it smarter.

Figure 2: Tutoring Summary Panel


Edit Lesson

Expectation

Expectation 1
You need to author the initial content

Hint

Hint 1
What do you need to make before you can test a dialog?

Hint 2
The tutor has to know want it to teach. What do you need to do for i

Hint 3
How does the tutor get the expectation and hints that it uses?

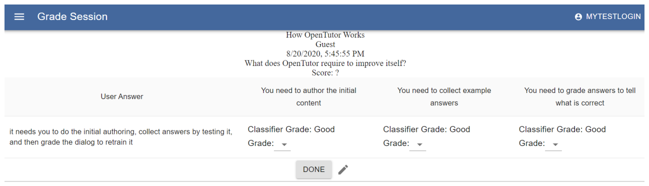
Figure 3: Lesson Editor - Expectations

Authors are then directed to visit the Grading List page, where the author rates the accuracy of the classifier in identifying good and bad answers. It is similar to the Lesson List except that it shows the list of tutoring sessions completed. By default, only ungraded sessions are shown, but all sessions can be displayed using a toggle. When a session is opened to grade (Figure 4), the user's answers to that session are shown in a table where each row is a user input and each column is an expectation that can be graded as good, bad, neutral, or left ungraded.

While all answers can be graded against all expectations, the session is counted as graded after each answer is graded against at least one expectation. There is a button to train a custom model for a lesson based on the grading, which is at the bottom of the lesson-editing screen but is disabled until each expectation has 5 answers graded *Good* and 5 graded *Bad*. This button starts online training and deploys a new model, which can be immediately tested when training completes successfully. Authors also see the cross-validation accuracy for the new model in the Lesson Editor UI.

Formative Usability Testing

A prototype of the OpenTutor authoring process was tested by five educators from domains other than computer science (e.g., mathematics, psychology, education). These educators were connected to universities (e.g., professors, instructors, graduate students with prior teaching experience)



How OpenTutor Works
Guest
8/20/2020, 5:45:55 PM
What does OpenTutor require to improve itself?
Score: ?

User Answer	You need to author the initial content	You need to collect example answers	You need to grade answers to tell what is correct
it needs you to do the initial authoring, collect answers by testing it, and then grade the dialog to retrain it	Classifier Grade: Good Grade:	Classifier Grade: Good Grade:	Classifier Grade: Good Grade:

Figure 4: Grading Interface

so they represent a relatively advanced group, in that all testers were highly experienced in traditional online courseware. Testers unfamiliar with the general concepts of an expectation-misconception dialog were given an introduction. Usability testing followed the process described above: log in, create a new lesson, complete a set of user sessions, grade the sessions, and initiate re-training the model. A 9-page tutorial was provided, with 12 large screenshots to demonstrate every major step. Authors had not previously used the tool. This first-dialog testing took between 50 and 80 minutes, including all instructions, authoring, grading, ratings, and discussion. At the time the usability testing was performed, the classifier training pipeline was still being debugged so authors did not retry their dialogs after training.

The instructors were encouraged to talk aloud and also rated the system on two items adapted from the Technology Acceptance Model (Venkatesh, Thong, and Xu 2016): ease of use and expected benefit. All ratings were on a six-point scale from 1=Definitely Not to 6=Definitely Agree (where 3=Slightly Disagree and 3=Slightly Agree). The ratings are shown in Table 1. Overall, the ratings were positive but not consistently strong: while OpenTutor was rated a good idea (concept), ratings of ease of use and expected benefit to students were both about a half-point lower. Of these, ease-of-use issues showed relatively actionable solutions while increasing benefit to students was more complex.

Ease-of-use ratings for individual editing panels were high, and primary concerns fell into three categories: transitions between different parts of the system (user experience flow) and better explanations for authored fields/values. The most common verbal comment (3 of 5) was that it was unclear how to launch a lesson to test it (particularly from the Lesson Editor). An equally common request was that elements of the Lesson Editor ("Introduction", "Hint") or Grader ("Good", "Bad") had a key or tooltips to help a new user know what to expect when they set those values, such as if an Introduction would be in chat from the agent versus shown in a panel before starting the dialog. Related to this, two users had trouble designing OpenTutor hints and instead posed yes-or-no hint questions. On the positive side, most authors praised the "clean dialog system" with the interfaces being easy to navigate and operate (e.g., "Working with the system and putting things together was intuitive").

The primary features requested by authors were greater multimedia/UI control and greater granularity in grading student responses. Multiple participants requested the ability to configure the UI shown to the student, such as uploading images or linking to segments of video clips. Some also wanted the ability to change the media shown based on dialog events (e.g., change an image after feedback). When

Item	Mean	StdDev
Overall, how good of an idea is OpenTutor as a tool to help learners?	5.2	0.5
How much do you think OpenTutor could be used to benefit your instruction and outcomes for students you are teaching?	4.6	1.1
Overall, was the system easy to use?	4.6	0.9
- Ease of Use: Home Screen/Listings	5	1.2
- Ease of Use: Lesson Editor	5	0.7
- Ease of Use: Grading Panel	5	1

Table 1: Non-CS Educator Ratings

grading student responses, authors found the current four options too limited (Good, Neutral, Bad, Ungraded). Requested additions took different forms, from a 5-point scale, to replacing Neutral with two tags for Partial or Mixed, or to split out a Does Not Address/Not Relevant label to use rather than just Bad.

The comments associated with expected benefit to students were more complex. The primary tradeoff was how often teachers would use this in a class versus items that were equivalently fast to author (e.g., multiple choice with feedback) but did not require grading to tune them. While one teacher indicated they would use this for a wide set of content, two others noted that they would only use this for deeper or more challenging content. However, for some types of complex content they were concerned if the AI would adequately understand student answers. As such, further research should survey teachers' preferred use cases for dialog-based tutoring in a larger course structure. Combined with research on where tutoring dialogs significantly outperform traditional methods, this could help develop guidelines for teachers to get the most value out of tutoring dialogs.

Conclusions

While OpenTutor is an actively-developing prototype, its design explores approaches that future authoring tools and dialog-based tutors. First, the high user acceptance for the authoring and grading interfaces indicate that teachers comfortable developing online courses can rapidly develop a first-run interactive tutoring dialog if provided UI designs that resemble more familiar tools provided by LMS and CMS frameworks. Second, the grading process was more efficient than expected, with authors typically able to grade sessions in under two minutes. As a result, the initial usability results were promising and may be relevant for future AI-based tools for instructors, where their grading activities might be used to train tools that help them with future class cohorts or activities. With that said, the sample size was small and not representative of typical instructors, so further research is needed to test feasibility for broader populations.

Additionally, from a comparative-advantage standpoint, instructors indicated that *any increase* in item authoring time (compared to multiple choice) would need to produce substantial student benefits, which could limit use to a subset of course lessons. We believe this can be addressed by considering two complementary directions: improved machine learning (decrease time costs) and multi-activity support (increase benefits). The OpenTutor approach should also be compared against established tools such as AutoTutor to better quantify efficiency improvements versus limitations that might reduce instructors expected benefit.

Improved machine learning for the dialog classifier is important, since every labeled example costs additional author time. The long-term goal for this classifier is to use every optimization possible to address the cold-start problem for new dialogs (e.g., through transfer learning, active learning, etc.). A core research problem for tutoring classifiers are that the correctness of an answer is often not entirely semantic, but depends on the context. For example, if an electronic circuit is shown for diode current flow, good answers may include: "in forward bias", "from left to right", "from anode to cathode", or even "in the direction of the arrow" (since a diode symbol looks like an arrow). An unsupervised general model such as Word2Vec is inherently incapable of directly understanding that these answers all demonstrate the same underlying concept (diode forward bias behavior).

As such, ongoing work is examining techniques to cluster user answers to find distinct patterns that imply semantically distinct ways to explain a concept. This direction is promising, because certain types of clustering are effective with small numbers of samples (e.g., hierarchical clustering, k-nearest-neighbors) and because research on human graders finds that the speed to grade answers can be more than doubled by using clustering to group answer patterns for short answers (Brooks et al. 2014). Extracting patterns from clusters should theoretically be more efficient than the n-gram keyword extraction techniques used by Nye et al. (Nye, Hajeer, and Cai 2015) which were determined from brute force (combinatorial) patterns based on exact keyword matching.

As a complementary approach, the benefits of OpenTutor could be enhanced by supporting alternate content modes, starting with a simple self-reflection survey or dialog. In this model, teachers could rapidly develop multi-part open ended questions (without assessment), which can then also convert into graded short answers or tutoring dialogs after sufficient sessions are graded. While this concept is still being explored, the ability to reuse the inferences from the lessons and classifier may offer a more broader feature set for teachers. As such, further work is needed to explore how dialog-based ITS fit into the larger content ecosystem for a course.

Acknowledgments

This research work was sponsored by Military Operational Medicine Research Program (MOMRP) conducted under the USC ICT University Affiliated Research Center (UARC; Army W911NF-14D-0005) and includes contributions from NSF REU student researchers (NSF IIS 1852583). However, all statements in this publication are the work of the authors alone and do not necessarily reflect the views of the sponsors, and no official endorsement should be inferred.

References

Brooks, M.; Basu, S.; Jacobs, C.; and Vanderwende, L. 2014. Divide and correct: using clusters to grade short answers at scale. In *Proceedings of the first ACM conference on Learning@ scale conference*, 89–98.

Dermeval, D.; Paiva, R.; Bittencourt, I. I.; Vassileva, J.; and Borges, D. 2018. Authoring tools for designing intelligent tutoring systems: a systematic review of the literature. *International Journal of Artificial Intelligence in Education* 28(3):336–384.

Graesser, A. C. 2016. Conversations with autotutor help students learn. *International Journal of Artificial Intelligence in Education* 26(1):124–132.

Kulik, J. A., and Fletcher, J. 2016. Effectiveness of intelligent tutoring systems: a meta-analytic review. *Review of educational research* 86(1):42–78.

Mikolov, T.; Sutskever, I.; Chen, K.; Corrado, G. S.; and Dean, J. 2013. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, 3111–3119.

Neil, K. 2018. Authoring interactive narrative in twine 2 vs ink vs yarn. https://medium.com/haikus_by_KN/authoring-interactive-narrative-in-twine-2-vs-ink-a-quick-and-dirty-comparison-using-examples-e695eb4dfc3e.

Nye, B. D.; Yang, M.; Hays, P.; Silva-Lugo, R.; Cai, Z.; Rahman, M. F.; Hu, X.; and Graesser, A. C. 2014. Rapid, form-based authoring of natural language tutoring trialogs. In *GIFTSym2*, 175–185.

Nye, B. D.; Graesser, A. C.; and Hu, X. 2014. Autotutor and family: A review of 17 years of natural language tutoring. *International Journal of Artificial Intelligence in Education* 24(4):427–469.

Nye, B. D.; Hajeer, M. H.; and Cai, Z. 2015. Improving classification of natural language answers to its questions with item-specific supervised learning. In *FLAIRS Conference*, 436–468.

Rus, V., and Lintean, M. 2012. An optimal assessment of natural language student input using word-to-word similarity metrics. In *International Conference on Intelligent Tutoring Systems*, 675–676. Springer.

Sultan, M. A.; Salazar, C.; and Sumner, T. 2016. Fast and easy short answer grading with high accuracy. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, 1070–1075.

Swartout, W.; Nye, B. D.; Hartholt, A.; Reilly, A.; Graesser, A. C.; VanLehn, K.; Wetzel, J.; Liewer, M.; Morbini, F.; Morgan, B.; et al. 2016. Designing a personal assistant for life-long learning (PAL3). In *FLAIRS 2016*, 491–496. AAAI Press.

Venkatesh, V.; Thong, J. Y.; and Xu, X. 2016. Unified theory of acceptance and use of technology: A synthesis and the road ahead. *Journal of the association for Information Systems* 17(5):328–376.

Wang, L.; Shubeck, K.; Shi, G.; Zhang, L.; and Hu, X. 2020. Cbits authoring tool in gift. In *Generalized Intelligent Framework for Tutoring (GIFT) Users Symposium (GIFTSym8)*, 69–77. US Army.

Wolfe, C. R.; Widmer, C. L.; Reyna, V. F.; Hu, X.; Cedillos, E. M.; Fisher, C. R.; Brust-Renck, P. G.; Williams, T. C.; Vannucchi, I. D.; and Weil, A. M. 2013. The development and analysis of tutorial dialogues in autotutor lite. *Behavior research methods* 45(3):623–636.