Automated Analysis of Student Verbalizations in Online Learning Environments

Nazik A. Almazova[1], Jason O. Hallstrom[1], Megan Fowler[2], Joseph Hollingsworth[3], Eileen Kraemer[2], Murali Sitaraman[2], and Gloria Washington[4]

[1] Florida Atlantic University, Boca Raton FL 33433, USA
[2] Clemson University, Clemson SC 29634, USA
[3] Rose-Hulman Institute of Technology, Terre Haute IN 47803, USA
[4] Howard University, Washington DC 20059, USA

1 Introduction

Online learning has become desirable for many students. In the U.S., more than one-third of all enrolled students participate in at least one online course [13]. The most effective online learning environments allow students to work at their own pace, from any location, at any time, and to receive automated feedback. In light of these benefits and the likely protracted impact of the current public health crisis, the trend toward online learning is likely to increase.

In traditional face-to-face learning environments, instructors adjust instruction based on student questions, specific misunderstandings, and attitudes (e.g., uncertainty, frustration). The limitation of many online learning environments is that they rely exclusively on objective student response data (e.g., a specific technical answer to a coding question) to base their feedback. This could result in less effective learning and students' giving preference to in-person instruction [19]. This research is about using more "noisy" data, student verbalizations, as a basis for personalized feedback.

While the ideas and results in this paper are broadly applicable to any online learning system, we have chosen to focus on an extension to an online environment for teaching code reasoning skills that our research team has developed over the past decade. In prior research, we have investigated how students perceive the tool, how well they understand the concepts taught, and the specific difficulties they encounter during various exercises. Achieving this understanding while students are using the tool (i.e., not just *post hoc*) is essential to being able to design *automated* interventions that help students overcome difficulties when they arise—not just for the reasoning tool, but for any online tool.

Objective assessments and attitudinal surveys are readily introduced in online environments, and these instruments help to reveal technical and attitudinal difficulties. However, these instruments are insufficient by themselves; they often fail to provide a high-fidelity view of the subtle technical and attitudinal difficulties students face when engaging in online problem-solving activities. Achieving this level of fidelity requires a view into students' internal cognitive processes.

A "think-aloud" protocol, which asks students to verbalize their thought processes during a problem-solving activity, offers a glimpse into student thinking. It can be powerful in illuminating technical and attitudinal problems that would otherwise be missed, but scalability is a problem. The protocol requires individual student sessions, manual transcription of audio data, and skilled in-

terpretation of results. The process is labor-intensive, and with typical resource shortages, cannot be applied across large student populations.

To overcome these scalability limitations, we present an automated approach to collecting and analyzing think-aloud data in online learning environments. The approach relies on an extension to web-based environments, which provides automated screen and audio recording, transcription, and labeling of specific characteristics of student attitudes (e.g., uncertainty), behaviors (e.g., guessing), and difficulties (e.g., logic misunderstandings). The implementation uses natural language processing and machine learning. The long-term goal of this labeling is to support automated interventions that are tailored to the specific misunderstandings student face, as well as their evolving emotional and attitudinal states, which have been shown to have a significant impact on learning. We evaluated the approach in two online sections of a required, junior-level algorithms course during the Spring of 2020 (n=18, n=13, respectively).

We expect the approach will improve online learning for a variety of learners, across subjects and topics. Here we focus on understanding the difficulties learners face when reasoning about code. Concretely, we focus on the following questions: ***ERQ 1:*** How accurate are existing transcription tools in transcribing domain-specific language, specifically, language associated with code reasoning? ***ERQ 2:*** How suitable are existing natural language processing and machine learning models to identify points of interest in student attitudes, behaviors, and cognitive processes for purposes of automating interventions?

2 Background and Related Work

Think-aloud Analysis. Analysis of think-alouds has found a variety of uses. In Nielson [29], think-alouds are considered among the set of usability inspection methods for evaluating user interfaces. Fan et al. [12] discuss the high value of using a think-aloud protocol for identifying and pinpointing users' problems and the cost of manual processing of think-aloud data. In [21], a real-time, in-class note-taking system for hearing-impaired CS students is discussed. Accuracy rates for real-time transcription range from 75% for an untrained system, to 94% for a trained system using a customized dictionary and customized pronunciations for 10 domain-specific words. The idea of detecting morphological forms of software-specific terms found in informal writing has also received attention. In [5], an automatic natural language processing approach is used for creating a thesaurus [7] from question and answer posts on forums such as Stack Overflow.

Role of Emotions and Guessing in Learning. The labels used in our verbalization analyzer are informed by the literature. Emotions, in particular, play an important role in learning [22, 31, 32, 41]. A number of groups have focused on identifying the links between learning and emotions [4, 10, 20, 23, 36], with application to improving the effectiveness of intelligent learning systems and providing personalized feedback [11, 15, 37]. Various emotions and affects have been considered (e.g., boredom, confusion, delight, surprise, frustration).

Bosch et al. consider the emotions experienced during the first programming experience and their correlation with performance [4], with a goal of automating

detection and adaptation in intelligent learning environments. Affective states most commonly observed in the study include *confusion/uncertainty*, *frustration*, *boredom*, and *neutral*. Not surprisingly, confusion/uncertainty at an early stage is associated with a significant decline in performance. In a related study, frustration (usually associated with a negative outcome) is found to be positively correlated with learning [30].

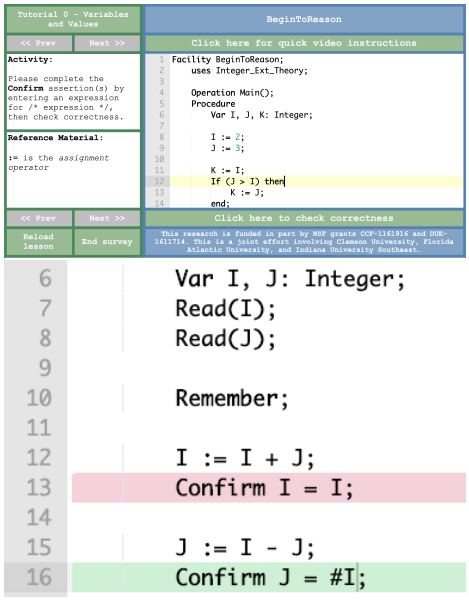

Fig. 1. BTR Tracing Activity

Girard et al. identify a similar set of labels for an affective learning companion in a meta-tutor project [15]. The *gaming* label identifies students who try multiple answers in a short period of time without allocating enough time to think about the problem (i.e., guessing). In [2], Baker et al. associate students "gaming the system" (i.e., guessing) with reductions in learning. The authors emphasize the importance of detecting and addressing guessing behaviors in intelligent learning environments.

Existing Tool Background. The experience reported here applies to a variety of online learning systems. The present experimentation has involved an extension to an online tool we have used to teach analytical reasoning that we use in our classrooms.

The online reasoning tool, BTR [35], aims to facilitate abstract code reasoning. Tracing and gaining code understanding based on *arbitrary* inputs (i.e., symbolic reasoning), instead of using one or two concrete inputs, is essential to becoming a strong programmer. Exposure to code reasoning basics, particularly through symbolic input values, provides further benefits since a variety of other concepts and skills become easier to grasp (e.g., in algorithms and data structures).

BTR includes a variety of activities, from code tracing on specific, concrete inputs; to symbolic reasoning over arbitrary inputs; to reasoning about objects, contracts, and invariants. The tool has been tested by over a thousand students over multiple years and courses. Results show that it is possible to help students learn the basics of abstract reasoning with minimal instruction [14], and it is possible to pinpoint key technical challenges students encounter based on directed activity paths and assessment of student response data [9].

BTR's student interface, shown in Figure 1, consists of *Activity*, *Reference Material*, and *Exercise* sections. The Activity section describes the current exercise, with instructions on how to complete it. The Reference Material section provides definitions for any mathematical symbols and/or operators used in the

exercise. The Exercise section displays the current exercise, instructing students to trace through a small fragment of Pascal-like code and to complete a logical `Confirm` assertion on the program variables, which tests their understanding.

After an answer is entered, *Check Correctness* is clicked to validate the student's answer. Importantly, the system relies on a verifying compiler [45] to validate the student's answer using mathematical verification; it does not rely on code execution or a prescribed list of student responses. The interface then provides immediate visual feedback [1], reducing cognitive load [27].

The example shown in Figure 1 asks students to assert the relationships between variables `I` and `J` at the two points marked `Confirm`, using `#I` and `#J` to refer to the values of these variables at the point marked `Remember`. Feedback is shown as red and green highlights, denoting incorrect and correct responses.

3 Online Think-Aloud Extension

As a first step towards building a verbalization analyzer and to pilot the think-aloud extensions, the tool was configured with 14 reasoning activities. The activities included tracing a concrete value assigned to a variable, swapping symbolic values of variables, finding the smallest and largest symbolic values given, finding the absolute value of a symbolic variable, and other similar exercises.

The think-aloud extension to BTR provides automated recording of screen and audio data, real-time transcription and logging, and automated tagging of transcribed speech with inferred characteristics of technical and attitudinal challenges expressed. When the tool is first loaded, students are presented with instructions on how to think out loud.

Before proceeding to the reasoning activities, the think-aloud extension requests access to the student's microphone and screen, and then runs a five-second recording test. The test may have to be repeated if there are problems, though for most users, one round of testing is sufficient. Once testing is complete and quality is ensured, the system enters the think-aloud state. When in the think-aloud state, the tool records audio and screen content, performing real-time transcription as the student works through the exercises.

The extension is built as a conventional web application, with an Amazon Web Services (AWS) back-end designed to support a large number of simultaneous think-aloud sessions. Media content is stored in an AWS S3 bucket, with transcribed speech stored in an AWS RDS instance to facilitate downstream analysis. Browser-based recording of screen and audio content relies on the *MediaStream Recording* API [44]. Recordings are captured in 30-second fragments and uploaded to AWS incrementally, limiting load.

While speech-to-text services have advanced significantly in recent years, high accuracy is only possible in perfect conditions. There are limitations in typical environments, stemming from background noise, recording equipment quality, muffled and/or quiet speech, and domain-specific technical terms—a key focus of this paper. We explored multiple transcription options, including Google's SpeechRecognition [17] interface and the DeepSpeech [28] library.

Google's SpeechRecognition interface is part of the Web Speech API, a W3C Community Specification [16], currently implemented only in Chromium-based

browsers (i.e., Chrome, Edge, Opera). The current implementation relies on the Google Cloud Speech-to-Text API. Benchmarking results show that the API achieves an error rate of 12.23% on LibriSpeech's test-clean dataset [34, 38]. While these results are impressive, it is important to note that they reflect performance on typical, conversational speech. Understanding whether they translate to domain-specific speech is a key goal of the work presented here.

DeepSpeech [28] is an open-source speech-to-text engine developed by Mozilla, which uses a TensorFlow-based deep learning algorithm. Benchmarking results show that the implementation achieves a 6.5% error rate on LibriSpeech's test-clean dataset [26]. These results are comparable to human performance [24] – but again reflect performance on conversational speech. Of note, DeepSpeech makes it difficult to achieve real-time transcription. In the current implementation of the verbalization analyzer, transcription via SpeechRecognition occurs in real-time, whereas transcription via DeepSpeech occurs post hoc.

4 Experimental Set Up

The experiments were conducted in a required third-year algorithms course at a Hispanic-Serving Institution. They were conducted in synchronous online settings via WebEx. Two sections of students were involved (n=18, n=13).

Each pilot began with a brief introduction to the goals of the study and the structure of the experiment. Students were introduced to the BTR tool and the think-aloud extension. The instructor demonstrated how to use the tool, how to complete a reasoning exercise, and how to think out loud. Students were encouraged to describe the steps in their thinking process while solving the exercises. The introduction was followed by the individual think-aloud sessions. Students were instructed to disconnect from the WebEx conference, and to use the web link to begin their think-aloud sessions. Each student completed a think-aloud session individually, from home, at their own computer.

5 ERQ 1: Transcription Accuracy

In the first pilot, students completed an average of 11 exercises out of a possible 14, with 9 students completing all exercises. Audio and video were captured per student, per session. The average recording length was 41.1 minutes per session, with a minimum length of 13 minutes, and a maximum length of 110 minutes. In the second pilot, students completed an average of 10 exercises, with 4 students completing all exercises. The average recording length was 29 minutes per session, with a minimum length of 9 minutes, and a maximum length of 46 minutes. Video content was used to study how students engaged with the tool, to support transcription correction, and to select the label set.

Audio recordings were processed using the Speech-Recognition and DeepSpeech implementations, generating per student, per session transcripts. All recordings were also manually transcribed to enable accuracy evaluation. The first pilot generated 18 transcripts, with an average word count of 1,562, a minimum word count of 288, and a maximum word count of 3,161. The second pilot generated 13 total transcripts, with an average word count of 1,269, a minimum

word count of 192, and a maximum word count of 2,605. In total, 31 per-session transcripts were collected across the two pilots.

WC	SR WER	DS WER	WC	SR WER	DS WER
288	0.51	0.59	1755	0.42	0.53
714	0.47	0.48	1756	0.53	0.57
718	0.12	0.21	1835	0.21	0.29
747	0.43	0.57	1889	0.36	0.49
810	0.35	0.48	1996	0.63	0.55
1114	0.33	0.44	2618	0.29	0.43
1423	0.59	0.63	2688	0.37	0.45
1487	0.34	0.49	3161	0.18	0.40
192	0.14	0.22	1257	0.23	0.32
397	0.34	0.39	1800	0.34	0.39
662	0.37	0.40	1837	0.39	0.49
716	0.37	0.44	1898	0.34	0.52
830	0.41	0.42	2446	0.43	0.58
861	0.43	0.56	2605	0.36	0.49
1005	0.23	0.33			

Table 1. Error Rate per Transcript, Across Pilots

Table 1 details the per-transcript word error rates (WER) for the two pilots, respectively (top-half=pilot 1, bottom-half=pilot 2). Overall, DeepSpeech exhibited an error rate of 45% (DS WER), and SpeechRecognition exhibited an error rate of 36% (SR WER). From these results, we conclude that at this time, neither state-of-the-art tool offers sufficient accuracy of transcription in the context of code reasoning.

Given the significantly better performance of SpeechRecognition and DeepSpeech in prior benchmarking studies, the relatively poor performance of these systems may be surprising. Comparisons between the manually collected and automatically collected transcripts show that a significant portion of the errors are due to limitations in handling references to the code text. Both systems were trained using conversational speech, presenting a context mismatch during transcription. Note that this limitation might be overcome using DeepSpeech, which supports training on a custom corpus. With training on a robust corpus collected in the context of code reasoning, performance is likely to improve.

6 ERQ 2: Identification Accuracy

We now turn our attention to identifying points of interest in student behaviors, attitudes, and difficulties. Given the error rates in the automatically generated transcripts, manually generated and validated transcripts are used hereafter.

Dataset Labeling. Labels corresponding to points of interest likely to be relevant in the design of automated interventions were identified through a careful literature review and manual analysis of audio, video, and transcript content. They are summarized in Table 2. Each of the 435 transcripts were manually labeled with the appropriate tags for testing purposes.

One important threat to validity is the subjective nature of the labeling process. Labels such as `uncertain` and `guessing`, for instance, are subject to interpretation, varying from one observer to another. To mitigate subjectivity, the label set was established iteratively through independent consideration of

a subset of transcripts by two independent labelers. After multiple iterations of independent labeling and comparison, the intent of each label was carefully described and documented, along with heuristics and examples. Subsequently, the transcript set was shuffled, and labels were independently assigned to all 435 transcripts. Inter-rater reliability was assessed using Cohen's kappa [8]. The agreement rate was high, with individual label agreement ranging between 0.84 and 1.0, with an overall agreement rate of 0.91. The final dataset used for analysis was created by merging the two independent datasets, with conflicting labels argued between the raters until consensus was reached.

Preprocessing and Vectorization. The input data for each student attempt includes a think-aloud transcript and BTR state (e.g., exercise number, attempt count, whether the exercise was solved correctly). Each response (i.e., answer) is encoded as a vector of tokens, including a boolean indicating correctness (i.e., 0 or 1). The vectors are padded, if necessary, to ensure consistency.

Label	Definition
`read_instructions`	The student read the exercise instructions out loud.
`traced_code`	The student traced the code out loud.
`uncertain`	The student was not confident in her reasoning.
`guessing`	The student was guessing, using brute-force.
`btr_confusion`	The student misunderstood a core BTR concept.
`logic_confusion`	The student misunderstood a core logic (programming) concept.
`intermediates_conf`	The student misunderstood how to differentiate and/or carry-through intermediate states across multiple, dependent statements.

Table 2. Attitude/Behavior/Misunderstanding Labels

Transcript data is preprocessed to remove low-value content. This includes removal of punctuation, symbols, single-letter words, numbers, and stop-words. We use NLKT's [3] default set of stop-words (e.g., "it", "its"). Lemmatization is also performed: Multiple forms of the same word are often used—for instance, multiple verb forms, such as *thinks* and *thinking*. It is often beneficial to reduce inflectional forms of a word to its root [18, 43]. Lemmatization is performed using NLTK's part-of-speech tagger and WordNet [42].

Most natural language labeling algorithms operate on *vectorized* text, collapsing each transcript to a vector of real numbers, or a set of such vectors, representing *features* of the text. We explore a range of vectorization techniques.

TF-IDF [40] computes a vector of weights for each transcript, where the length of each vector is equal to the number of words contained in the corpus (i.e., all preprocessed transcripts). The weight for each word in the vector for a specific transcript is calculated as the word's frequency in the transcript, divided by its frequency in the corpus. (Words that are rare in the corpus, but appear

frequently in a transcript, receive larger weights in that transcript's vector.) Note that the approach ignores information about word position. A phrase like, "I don't understand" is treated as a collection of three words, not a single concept. To overcome this limitation, we consider TF-IDF using phrase lengths of one to three words (i.e., *n-gram* lengths of one to three).

Word2Vec [25] is a *word embedding* algorithm that relies on a neural network model to generate a vector of real numbers for each word in the corpus. The vectors are intended to capture relationships among words, so that, for instance,"don't understand" - "don't" = "understand". The mechanics of the neural network are omitted here, but they rely on the relative context of words within phrases across the corpus. A Word2Vec model was trained on the think-aloud corpus, using a specified vector dimension of 100. This resulted in a set of vectors for each transcript, containing one vector for each word in the transcript.

GloVe [33] is another embedding algorithm intended to capture the relationships among words. The mechanics, again omitted, rely on word-word co-occurrence statistics. While a custom GloVe model can be trained, we use the pre-trained model, which benefits from a large corpus. Whereas our domain-specific corpus includes 10,898 tokens (total words), and a vocabulary of 772 distinct words, the GloVe corpus includes 840B tokens and a vocabulary of 2.2M words. Encoding using the pre-trained model generated a vector of dimension 300 for each word in the corpus and corresponding sets of vectors for each transcript. While all three methods were evaluated in the context of the labeling algorithms described next, only GloVe performed significantly better than 50% labeling accuracy. It remains the focus hereafter.

7 Data Analysis and Results

The binary relevance method [46] is one of the most common techniques for approaching multi-label classification. The problem is decomposed into seven sub-problems, one for each label (i.e. positive or negative). As detailed next, a variety of models were explored to automate label assignment. A key factor in achieving high accuracy when training these classifiers is the size of the training set (a portion of our transcript set) and the distribution of labels across the set. Labels that are underrepresented or overrepresented can bias the classifier's performance. Even at 435 per-attempt transcripts, our training sets are relatively small. More importantly, as illustrated in Figure 2, the label distributions are not balanced. The `guessing` label, for instance, is assigned to only 47 transcripts, whereas the `traced_code` label is assigned to 265 transcripts.

To achieve balance across labels, weight adjustments were applied during training. While the exploratory results are omitted here, the balancing approach improved the accuracy of label assignment by as much as 30% for some labels. The labeled, vectorized dataset was shuffled and split, with 70% of the transcripts used for training, 10% withheld for validation and 20% withheld for testing. The split was stratified across labels to ensure balanced representation.

7.1 Classifier Architecture and Model Training

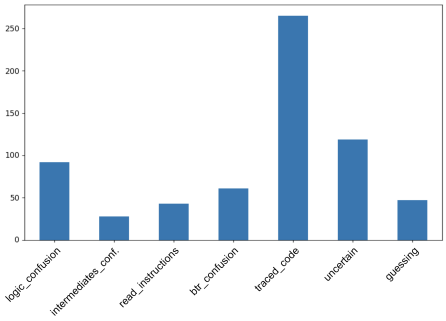

Fig. 2. Dataset Label Distribution

Several neural network architectures were evaluated, all with previously demonstrated success in text classification problems, including fully connected neural networks, convolutional neural networks, recurrent neural networks (RNN) with gated recurrent units, and long short-term memory (LSTM) networks [39]. The open-source Keras library [6] was used for network implementation and evaluation. The best performing architecture is structured as follows.

The first input layer accepts a tokenized transcript (i.e., a set of integers corresponding to the corpus vocabulary), which is passed to a pre-trained embedding layer. The embedding layer transforms the integer set into a GloVe-based embedding representation (i.e., a set of embedding vectors of dimension 300). The embedding layer is followed by two LSTM networks [39], recurrent structures designed for problems involving order-dependence. An interspersed dropout layer limits network over-fitting, enhancing generalizability. The second input layer (dimension=3) accepts exercise number, attempt count, and attempt outcome. The third input layer (dimension=66) accepts the tokenized solution entered by the student. A concatenation layer combines the three inputs into a single tensor. A fully connected layer of 32 units with rectified linear activation (ReLU) trains on all data combined. Finally, a fully-connected output layer with one sigmoid activation function outputs a likelihood score (0.0-1.0) corresponding to the label result. A score at or above 0.5 corresponds to *true*, and lower values correspond to *false*. This architecture was trained to generate seven different models, corresponding to each of the labels, respectively. (Details concerning hyper-parameter tuning for each model are omitted.) All models required fewer than 1,000 epochs to converge. The best performing models, with minimal loss scores across both the training and validation sets were used.

Label	Acc.	n	Label	Acc.	n	Label	Acc.	n
`uncertain`	88.5%	26	`btr_confusion`	87.8%	15	`traced_code`	87.0%	56
`logic_conf.`	80.2%	20	`read_inst.`	92.4%	12	`inter._conf.`	96.2%	10
`guessing`	94.7%	10						

Table 3. Automated Labeling Accuracy (per label)

Table 3 summarizes labeling accuracy over the withheld test set (n=87), as well as the number of transcripts with each label. There were 20 transcripts corresponding to the `logic_confusion` label, for instance, and these were identified correctly 80.2% of the time – the lowest across the labels. There were 10 transcripts corresponding to students who had `intermediates_confusion`, and these were correctly identified 96.2% of the time – the highest accuracy across the labels. Of course, the significance of the results varies from one label to an-

other due to variation in representation within the test set – but overall, the classifier performed well, with an average accuracy of 89.5%

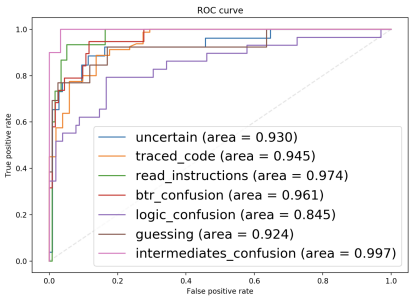

Fig. 3. ROC Curves for All Models

Recall that each of the seven trained networks outputs a likelihood score (0.0-1.0) for each transcript, with a discrimination threshold of 0.5. The *receiver operating characteristic* (ROC) plot shown in Figure 3 is generated by varying this threshold to explore the trade-off between the true positive rate and the false positive rate. Increasing the discrimination threshold to 1.0, for instance, guarantees a 100% true positive rate, but at the cost of false positives. The area beneath each curve, shown parenthetically in the legend, is a measure of the network's discriminatory power – the likelihood the classifier will rank positive cases above negative cases. The discriminatory performance is strong, with an average area-under-curve of 0.94.

8 Conclusion

This paper presented results in automating the analysis of student verbalizations in online learning environments, using an existing code reasoning tutor as the study context. A new extension automatically transcribes student verbalizations and uses machine learning to identify specific student attitudes, behaviors, and difficulties. We presented an analysis of transcription accuracy and utility in identifying seven learner characteristics of interest. The results were based on experimentation in two online sections of a required, junior-level algorithms course, resulting in 435 think-aloud transcripts.

Unfortunately, the accuracy offered by current transcription tools appears to be insufficient to enable reliable transcription in the context of domain-specific verbalizations (e.g., code reasoning), hindering a fully automated implementation. However, this is only a temporary problem, almost certainly due to the absence of sufficient training on an appropriately robust, domain-specific verbalization corpus. Developing such a corpus is only a matter of time – and looking to the future, there is reason to invest in that effort. When transcription accuracy is high, our results provide evidence that a broad set of interesting characteristics concerning student attitudes, behaviors, and difficulties can be identified automatically, with high accuracy, in real-time. Enabling automatic identification of student difficulties would have a profound impact on the design and efficacy of online tutoring systems, in computer science education and beyond.

References

1. R. Azevedo and R. M. Bernard. A meta-analysis of the effects of feedback in computer-based instruction. *Journal of Educational Computing Research*,

13(2):111–127, 1995.
2. R. S. Baker, A. T. Corbett, K. R. Koedinger, and A. Z. Wagner. Off-task behavior in the cognitive tutor classroom: when students" game the system". In *Proceedings of SIGCHI*, pages 383–390, 2004.
3. S. Bird, E. Klein, and E. Loper. *Natural language processing with Python: analyzing text with the natural language toolkit.* " O'Reilly Media, Inc.", 2009.
4. N. Bosch, S. D'Mello, and C. Mills. What emotions do novices experience during their first computer programming learning session? In *International Conference on Artificial Intelligence in Education*, pages 11–20. Springer, 2013.
5. C. Chen, Z. Xing, and W. Ximing. Unsupervised software-specific morphological forms inference from informal discussions. In *The 39th International Conference on Software Engineering, Buenos Aires, Argentina*. IEEE, 2017.
6. F. Chollet et al. Keras, 2015.
7. C. Chunyang. Sethesaurus: synonyms and abbreviations for software-related terms.
8. J. Cohen. A coefficient of agreement for nominal scales. *Educational and psychological measurement*, 20(1):37–46, 1960.
9. M. Cook, M. Fowler, J. O. Hallstrom, J. E. Hollingsworth, T. Schwab, Y.-S. Sun, and M. Sitaraman. Where exactly are the difficulties in reasoning logically about code? experimentation with an online system. In *Proceedings of the 23rd Annual ACM Conference on Innovation and Technology in Computer Science Education*, pages 39–44, 2018.
10. S. Craig, A. Graesser, J. Sullins, and B. Gholson. Affect and learning: an exploratory look into the role of affect in learning with autotutor. *Journal of educational media*, 29(3):241–250, 2004.
11. L. S. Doddannara, S. M. Gowda, R. S. d Baker, S. M. Gowda, and A. M. De Carvalho. Exploring the relationships between design, students' affective states, and disengaged behaviors within an its. In *International Conference on Artificial Intelligence in Education*, pages 31–40. Springer, 2013.
12. M. Fan, J. Lin, C. Chung, and K. N. Truong. Concurrent think-aloud verbalizations and usability problems. *ACM Trans. Comput.-Hum. Interact.*, 26(5), July 2019.
13. N. C. for Education Statistics. Trend generator (student enrollment).
14. M. Fowler, M. Cook, K. Plis, T. Schwab, Y.-S. Sun, M. Sitaraman, J. O. Hallstrom, and J. E. Hollingsworth. Impact of steps, instruction, and motivation on learning symbolic reasoning using an online tool. In *Proceedings of the 50th ACM Technical Symposium on Computer Science Education*, pages 1039–1045, 2019.
15. S. Girard, M. E. Chavez-Echeagaray, J. Gonzalez-Sanchez, Y. Hidalgo-Pontet, L. Zhang, W. Burleson, and K. VanLehn. Defining the behavior of an affective learning companion in the affective meta-tutor project. In *International Conference on Artificial Intelligence in Education*, pages 21–30. Springer, 2013.
16. W. C. Group. Web speech api.
17. W. C. Group. Web speech api the speechrecognition interface.
18. P. Han, S. Shen, D. Wang, and Y. Liu. The influence of word normalization in english document clustering. In *2012 ieee international conference on computer science and automation engineering (csae)*, volume 2, pages 116–120. IEEE, 2012.
19. N. Kemp and R. Grieve. Face-to-face or face-to-screen? undergraduates' opinions and test performance. *Frontiers in Psychology*, 5:1278, 2014.
20. I. A. Khan, R. M. Hierons, and W. P. Brinkman. Mood independent programming. In *Proceedings of the 14th European conference on Cognitive ergonomics: invent! explore!*, pages 269–272, 2007.
21. R. Kheir and T. Way. Inclusion of deaf students in computer science classes using real-time speech transcription. *SIGCSE Bull.*, 39(3):261–265, June 2007.

22. L. Knörzer et al. Facilitators or suppressors: Effects of experimentally induced emotions on multimedia learning. *Learning and Instruction*, 44:97–107, 2016.
23. B. Kort, R. Reilly, and R. W. Picard. An affective model of interplay between emotions and learning: Reengineering educational pedagogy-building a learning companion. In *Proceedings IEEE International Conference on Advanced Learning Technologies*, pages 43–46. IEEE, 2001.
24. R. P. Lippmann. Speech recognition by machines and humans. *Speech communication*, 22(1):1–15, 1997.
25. T. Mikolov, K. Chen, G. Corrado, and J. Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.
26. R. Morais. A journey to ¡10% word error rate.
27. R. Moreno. Decreasing cognitive load for novice students: Effects of explanatory versus corrective feedback in discovery-based multimedia. *Instructional science*, 32(1-2):99–113, 2004.
28. Mozilla. Project deepspeech.
29. J. Nielsen. *Usability Engineering*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1993.
30. Z. A. Pardos, R. S. Baker, M. O. San Pedro, S. M. Gowda, and S. M. Gowda. Affective states and state tests: Investigating how affect throughout the school year predicts end of year learning outcomes. In *Proceedings of the third international conference on learning analytics and knowledge*, pages 117–124, 2013.
31. B. Park, J. L. Plass, and R. Brünken. Cognitive and affective processes in multimedia learning, 2014.
32. R. Pekrun and L. Linnenbrink-Garcia. *International handbook of emotions in education*. Routledge, 2014.
33. J. Pennington et al. Glove: Global vectors for word representation. In *Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, 2014.
34. Picovoice. Speech-to-text benchmark.
35. RESOLVE Software Research Group at Clemson University. BeginToReason.
36. M. M. T. Rodrigo and R. S. Baker. Coarse-grained detection of student frustration in an introductory programming course. In *Proceedings of the fifth international workshop on Computing education research workshop*, pages 75–80, 2009.
37. K. D. Sidney, S. D. Craig, B. Gholson, S. Franklin, R. Picard, and A. C. Graesser. Integrating affect sensors in an intelligent tutoring system. In *Affective Interactions: The Computer in the Affective Loop Workshop at*, pages 7–13, 2005.
38. O. Speech and L. Resources. Librispeech asr corpus.
39. R. Staudemeyer and E. Morris. Understanding lstm–a tutorial into long short-term memory recurrent neural networks. *arXiv preprint arXiv:1909.09586*, 2019.
40. tfidf.com. Tf-idf :: A single-page tutorial - information retrieval and text mining.
41. E. Um, J. L. Plass, E. O. Hayward, B. D. Homer, et al. Emotional design in multimedia learning. *Journal of educational psychology*, 104(2):485, 2012.
42. P. University. About wordnet.
43. A. K. Uysal and S. Gunal. The impact of preprocessing on text classification. *Information Processing & Management*, 50(1):104–112, 2014.
44. W3C. Mediastream recording.
45. D. Welch, C. T. Cook, Y. Sun, and M. Sitaraman. A web-integrated verifying compiler for RESOLVE: a research perspective. In D. Janakiram, K. Sen, and V. Kulkarni, editors, *7th India Software Engineering Conference, Chennai, ISEC '14, Chennai, India - February 19 - 21, 2014*, pages 12:1–12:6. ACM, 2014.
46. M.-L. Zhang, Y.-K. Li, X.-Y. Liu, and X. Geng. Binary relevance for multi-label learning: an overview. *Frontiers of Computer Science*, 12(2):191–202, 2018.