Just a Few Expert Constraints Can Help: Humanizing Data-Driven Subgoal Detection for Novice Programming

Samiha Marwan, Yang Shi, Ian Menezes, Min Chi, Tiffany Barnes, Thomas W. Price
North Carolina State University, Raleigh, NC, USA
samarwan, yshi26, ivmeneze, mchi, tmbarnes, twprice@ncsu.edu

ABSTRACT

Feedback on how students progress through completing subgoals can improve students' learning and motivation in programming. Detecting subgoal completion is a challenging task, and most learning environments do so either with *expert-authored* models or with *data-driven* models. Both models have advantages that are complementary – expert models encode domain knowledge and achieve reliable detection but require *extensive authoring efforts* and often cannot capture all students' possible solution strategies, while data-driven models can be easily scaled but may be less accurate and interpretable. In this paper, we take a step towards achieving the best of both worlds – utilizing a data-driven model that can intelligently detect subgoals in students' correct solutions, while benefiting from human expertise in editing these data-driven subgoal rules to provide more accurate feedback to students. We compared our hybrid "humanized" subgoal detectors, built from data-driven subgoals modified with expert input, against an existing data-driven approach and baseline supervised learning models. Our results showed that the hybrid model outperformed all other models in terms of overall accuracy and F1-score. Our work advances the challenging task of automated subgoal detection during programming, while laying the groundwork for future hybrid expert-authored/data-driven systems.

Keywords

Subgoals, Formative feedback, Data-driven hybrid models

1. INTRODUCTION

Formative feedback has been shown to be an effective form of automated feedback that can improve students' learning and motivation [54, 38, 8, 20]. In programming, *immediate* formative feedback *during* problem-solving is important because some problems require students to find one of many correct solutions [16], and novices may be uncertain about when they are on the right track [62]. This uncertainty may lead some students to give up [43], and can also negatively impact student confidence and motivation in computer science (CS) [32]. Prior research has shown that immediate feedback can address this, reducing novices' uncertainty and improving their confidence, engagement, motivation, and learning [42, 8, 33, 21, 38, 39].

One effective form of immediate feedback is subgoal feedback [40], which indicates students' progress on specific sub-steps of the problem. Feedback on subgoals offers special advantages because it demonstrates how a student can *break down a problem into a set of smaller sub-tasks*; which is a key to simplifying the learning process [37, 38], and can promote students' retention in procedural domains [35]. To generate such feedback, learning environments need to be able to do *subgoal detection*, which is the process of detecting when a student completes a key objective or sub-part of a programming task (e.g. receiving and validating user input). However, subgoal detection *during* problem-solving is known to be extremely challenging because it requires assessing students' *intended problem solving approach* rather than their *program output*. In other words, it is difficult to automatically evaluate whether a student completed a subgoal *in the middle* of problem-solving due to the many possible strategies that students can approach to solve a problem, even when using test cases or autograders.

Historically, to provide feedback on subgoals, learning environments have used expert-authored models, where human experts encode a set of rules to predict solution strategies that students might perform to complete a specific subgoal. While expert models can generate accurate feedback with interpretable explanations, they also require extensive human participation particularly for open-ended programming tasks, where it becomes unmanageable to capture every possible correct solution [59]. More recently, data-driven (*DD*) models, where the model learns rules from historical data, have become more prominent models. This is because *DD* models reduce the expert-authoring burden, and have the potential to be easily scaled to more problems and contexts. Moreover, *DD* models learn from multiple students' solutions, which makes it capture code situations that human experts cannot easily perceive, particularly in open-ended programming tasks [49, 45]. However, *DD* models are dependent on the quality of the data, and may have lower accuracy than expert models [59, 48]; and, therefore, may sometimes provide misleading feedback in practice [48]. Both of these models have strengths and weaknesses, and in this paper we propose an approach that takes advantage of both.

We present a hybrid approach that leverages both a data-driven model and expert insights to detect subgoal completion during problem-solving block-based programs. Our hybrid model is based on three main steps. First, we used an unsupervised data-driven model to generate subgoal detectors for a programming task, and represent them as a set of human-*interpretable* and human-*editable* rules. Second, this representation allowed us to evaluate the accuracy of the subgoal detectors; particularly when they have inaccurate detections. Third, we used human expert insights to refine and fix rules that led to inaccurate subgoal detections. These three steps resulted in a *hybrid* data-driven approach that generates subgoal detectors with high accuracy, that target expert-authoring effort *only* where improvements are needed, and that can also be easily scaled to various problems and contexts.

We evaluated our hybrid data-driven model to a block-based programming problem from an introductory CS classroom against the same, fully data-driven (*DD*) model, *but* without experts' intervention. We evaluated the accuracy of both models by comparing their subgoal detections on a given programming task, to that of human experts, and we hypothesize that our hybrid model will surpass the accuracy achieved by the *DD* model. We found that the expert evaluations of subgoal detections achieved significantly higher agreement with our hybrid model than that achieved with the *DD* model. We also found that our hybrid model outperforms the state-of-the-art supervised models: code2vec [53], Support Vector Machine (SVM) [14], and XGBoost [9]. In addition, we present case studies of how the hybrid model led to differing subgoal detections in student programs compared to the *DD* model. We also discuss how we can close the loop by applying our hybrid model in block-based programming classrooms to provide students with immediate feedback on subgoals.

In summary, in this paper we investigate this research question: **RQ:** *How well does a hybrid data-driven model combined with expert insights perform compared to: 1) a data-driven model without expert augmentation and 2) baseline supervised learning approaches that leverage expert subgoal labels?* Our work provides the following contributions: (1) we present a hybrid subgoal detection approach which combines an unsupervised data-driven model with domain expertise to achieve the benefits of both data-driven models, and expert models, and (2) we demonstrate how our hybrid approach advances the state of the art in subgoal detection in open-ended programming tasks over supervised and unsupervised baselines, with an accuracy range of 0.80 - 0.92.

2. LITERATURE REVIEW

In this work, we investigate the challenge of automatically detecting subgoals effectively. We propose a method that involves a hybrid approach where human experts modify data-driven models to build effective subgoal detectors. In the following, we first review prior work on the immediate feedback with a focus on subgoal feedback. Then, we review prior work that involves merging machine and human expert intelligence to improve performance of machine learned models. Finally, we review both state-of-the-art supervised learning models and an unsupervised data-driven model that we used for subgoal detection in a programming task.

2.1 Feedback and Subgoal Detection

Formative feedback is defined as a type of task-level feedback that provides *specific*, *timely* information to a student in response to a particular problem, based on the student's current ability [54]. In a review of effective formative feedback in education, Shute shows that immediate formative feedback is effective because it can improve students' learning [11, 38] and retention [54, 44], particularly in procedural skills such as programming [54]. Most intelligent tutoring systems provide immediate feedback through identifying errors (e.g. error detectors [5, 55], anomaly detectors [31], or misconception feedback [25, 24]); however far less work has been devoted to providing feedback on students' *subgoals*. Automated assessment systems (i.e. autograders) can provide feedback on correct subgoals by showing the *passing* test cases using expert-authored models [7, 28, 26, 38]. For example, most autograders use instructor test cases to check for correct program output; however they require students to submit an almost-complete program to obtain feedback [7, 29]. As a result, this feedback is often not available in the early stage of programming when timely feedback on subgoals is mostly needed. To provide timely subgoal feedback, prior research has taken two exclusive approaches: *expert-authored* approach and *data-driven* approach.

Expert-authored Approaches: Prior work has explored student completion to subgoals by diagnosing students' solutions against expert models (e.g. constraint-based models [42]), even when a student has incomplete submissions, to provide feedback on whether they are on track [27], or whether they completed key objectives of short programming tasks [38]. However, these systems often require extensive expert effort to create rules. To address this authoring burden, example-tracing tutoring systems infer tutoring rules based on examples of potential student behaviors. This still requires an author with some domain expertise, but it allows rules to be constructed by non-programmers who have domain expertise [2]. An expert can create different example solutions to capture different solution strategies; and augment them with hints or feedback. Example-tracing tutors have been developed in multiple non-programming domains like genetics [12], mathematics [1], and applied machine learning, and they have been shown to improve the problem-solving process and student learning [2]. Despite the accuracy of expert models in providing feedback on test cases or correct features, which can be equivalent to subgoals, it is unclear how feasible they are in domains with vast solution spaces and open-ended problems, such as in programming tasks [59, 39, 25].

Data-driven Approaches: Data-driven approaches refers to systematically collecting and analyzing various types of educational data, to guide a range of decisions to help improve the success of students [15, 50]. Data-driven models largely avoid the need for expert authoring altogether by using prior students' correct solutions, instead of expert rules or instructor solutions, to learn patterns of correct solutions. This enables automated assessment feedback on student code [21]. Many data-driven models work by executing a comparison function that calculates the distance between students' code and all the possible correct solutions, and then compares the path of the most close solution with that of the student [47, 49, 59, 39]. While most data-driven methods have

been used to generate fine-grained feedback – such as hints in the hint factory [58], little work has used these methods to generate subgoal feedback. For example, Diana et al. developed a model that searches for meaningful code chunks in student code to generate data-driven rubric criteria to automatically assess students' code [19]. Diana et al show that a data-driven model can have agreement with that of the experts [19]. In the iList tutor, Fossati et al. used a data-driven model to provide feedback on correct steps, where they assess student code edits as *good*, if it improves the student's probability to reach the correct solution, or *uncertain* if a student spent more time than that taken by prior students at this point [21]. Fossati et al. found that this feedback was well-perceived by students, and improved their learning [21]. However, data-driven models are dependent on the similarity of the current student's approach to prior student submissions, making it difficult to control the quality of their feedback [39, 51, 59, 46]. In a case study paper, Shabrina et al. discuss the practical implications of data-driven feedback on subgoals, showing that the quality of feedback is important, even positive feedback, since inaccurate feedback can cause students to spend more time on a task even after they were done [51]. Because of such challenges and perhaps other reasons such as the inability for individual instructors to augment autograder feedback, few tools have been built to provide immediate feedback to students on whether they have achieved subgoals during their programming tasks [34].

2.2 Integrating Expert Knowledge into Models

In recent years, combining machine and human intelligence has been extensively explored in a wide range of domains including artificial intelligence and software engineering. For example, Chou et al developed a virtual teaching assistant (VTA) that uses teachers' answers as human intelligence, and machine intelligence to use teachers' answers to locate student errors, and generate hints [10]. Chou et al found that these mechanisms reduce teacher tutoring load and reduce the complexity of developing machine intelligence [10]. In the software engineering domain, there is an emerging approach called *collective intelligence* that merges the wisdom of multiple developers with program synthesis algorithms, which has been shown to significantly improve the efficiency and accuracy of program synthesis [61].

Human-in-the-loop methods are another effective trend that use human intelligence to improve the efficiency of Machine Learning models, *while* the model is learning [23, 56, 30, 63]. For example, Goecks introduces a theoretical foundation that incorporates human input modalities, like demonstration or evaluation, to leverage the strengths and mitigate the weaknesses of reinforcement learning (RL) algorithms [23]. Their results show that using human-in-the-loop methods accelerates the learning rate of RL models, with a more efficient sample, in real time [23]. Our work also uses human intelligence to improve the accuracy of a machine learning model; however, it does so *after* the model is trained.

In the educational domain, expert knowledge is widely applied to augment data-driven and machine-learned models for problem solving and feedback. For example, in a logic tutor that provides data-driven hints using students' solutions, Stamper et al. used an initial small amount of sample data generated by human experts to enhance the automatic delivery of hints [57]. Moreover, example-tracing tutors allow experts to specify moderately-branching solutions for open-ended problems, allowing some intelligent tutors *originally* implemented using complex expert systems to be almost completely replicated to support practical learning needs [2].

2.3 Supervised Learning Models of Code Analysis

Supervised learning algorithms leverage labels created by human experts, to guide the model search process. With available labels, automated learning algorithms can be applied to the subgoal detection tasks for programming data. As shown in [6], one baseline is to extract term frequency-inverse document frequency (TF-IDF) features and uses traditional machine learning algorithms such as support vector machines (SVM) [14] and XGBoost [9]. However, as word- or token-based features such as TF-IDF lose important structural information from programming data [3], recent work uses structural representations from code and a more complex model structure to learn more complex features. For example, Shi et al. applied a code2vec [4] model to detect the completion of rubrics on student programming data [53]. In this work, we compared our hybrid data-driven model to these existing supervised learning baseline models to check our improvement on the subgoal detection task.

2.4 Data-Driven Subgoal Detection Model

Among the various data-driven models for detecting subgoals, or rubric items [39, 18, 19], we built our proposed hybrid model on top of an unsupervised data-driven subgoal detection (*DD*) algorithm, presented in [64]. We applied this algorithm on a programming task called *Squiral* (described in Section 3) by running the following steps. First, the algorithm extracts prior student solutions in *Squiral* as abstract syntax trees (ASTs) [49, 47]. Then, it applies hierarchical code clustering for feature extraction and selection by: (1) extracting common **code shapes**, which are common subtrees, in ASTs of correct students' solutions (Figure 2 shows examples of code shapes); (2) filtering redundant code shapes; (3) identifying decision shapes, which consist of a disjunction of code shapes (i.e. $C_1 \wedge \ldots \wedge C_n$) that are usually mutually-exclusive (e.g. a program uses a loop, or a repeated set of commands, but not both), and (4) hierarchically clustering frequently co-occurring code shapes into *subgoals*. In [64], the authors found a meaningful Cohen's Kappa (0.45) in agreement of the algorithm and expert subgoal detection on student data, suggesting that *DD* subgoals could be used to generate feedback. However, since the *DD* subgoals are typically represented in a regular-expression-like form, labels are needed to make them meaningful and usable for students in programming environments.

3. METHOD

This work presents and evaluates a hybrid data-driven model for generating and detecting subgoals in a block-based programming exercise (explained in Section 3.1). To evaluate our hybrid data-driven approach, we applied our model on student data collected from an Introduction to Computing course, and we asked human experts to evaluate the accuracy of its subgoal detections (explained in Section 3.2.2).

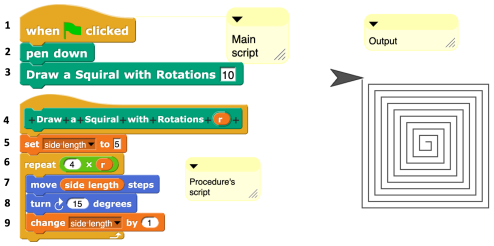

Figure 1: One possible solution for Squiral with line numbers on the left, and the script's output on the right.

We use this dataset to provide examples of how our approach works, but we also discuss how it can be generalized. We compared our hybrid model against its underlying data-driven (DD) model described above in Section 2.4, as well as state-of-the-art supervised learning models (explained below in Section 4.1).

3.1 Dataset

Our data is collected from a CS0 course for non-majors in a public university in the United States that uses Snap*!*, a block-based programming environment [22]. This programming environment logs all students actions while programming (e.g. adding, deleting or running blocks of code) with the *time* taken for each step. These student logs (i.e. actions) can also be replayed as a *trace* of code snapshots of all students' edits – allowing us to detect the time and the code snapshot where a subgoal is completed *during* student *problem-solving process*.

In this work, we collected students' logs from one homework exercise named *Squiral*, derived from the BJC curriculum [22]. *Squiral* requires a visual output, where students are asked to write a procedure that takes one input `'r'` to draw a square-like spiral with `r` rotations. Figure 1 shows a possible correct solution of *Squiral* that requires procedures, variables, and loops. We collected a training dataset from 3 semesters: Spring 2016 (S16), Fall 2016 (F16), and Spring 2017 (S17), which includes data of 174 students, that has a total log data of 29,574 student actions

3.2 Hybrid Data-Driven Subgoal Detection

Our hybrid data-driven model is based on two main things. First, the *DD* model is used to generate data-driven subgoals. Second, expert-authored constraints are added to improve the quality of these subgoals and the accuracy of their detection. We implemented our hybrid approach by conducting the following 3 high-level steps:

1. We used the *DD* model to generate an initial set of subgoals, consisting of clusters of code shapes. We then presented these clusters to experts in an interpretable form, who *combined* them to create a concrete set of meaningful subgoal labels.

2. We integrated *DD* subgoal detection model into the students' programming environment, allowing students to see when the *DD* algorithm detected completion of each subgoal. We then collected student programming log data along with *DD* detections, and asked experts to evaluate the accuracy of the *DD* detections.

3. We used human expert insights to fix code shapes that led to false subgoal detections; and then combined them again to create a modified set of hybrid subgoals and evaluated its new accuracy.

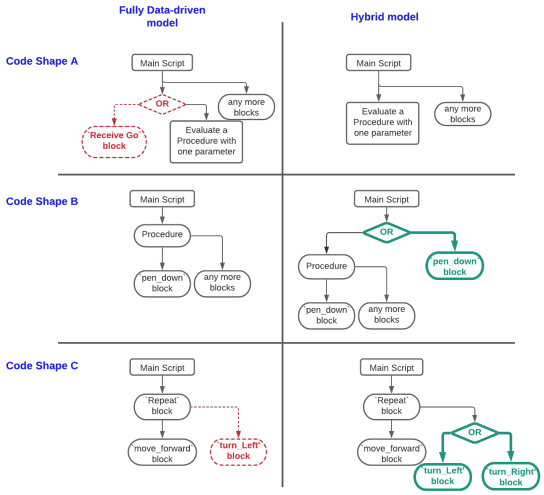

Figure 2: Three code shapes A, B, C, in both the data-driven and hybrid models. Each code shape represents a false detection and its fix by human experts. Red dashed nodes are removed and green bold nodes are added.

3.2.1 *Step 1: Interpreting and Editing Data-Driven Subgoals*

The goal of this step is to generate data-driven subgoals using the *DD* model and present them in an interpretable and editable form. We applied the *DD* model (described in Section 2.4) on S16, F16, and S17 student datasets to generate a number of clustered code shapes. Column 1 in Table 1 shows the description of 7 subgoals corresponding to code-shape clusters generated from *correct* solutions (n = 52). We evaluated each cluster by displaying its code shapes separately and *interpreted* their code behavior. For example, code shape A in Figure 2, on the left, represents a decision shape that requires student to use the `'ReceiveGo'` block (i.e. the hat block in Figure 1, line 1, which is used to start a script) in their main script, **or** to evaluate a procedure with one parameter, which is done by creating and snapping a procedure in the main script (as shown in Figure 1, line 3). We treated each cluster as a subgoal, and for a subgoal to be detected, the *DD* model requires all of its code shapes, and exactly one component of its *decision* shapes, to be present in student code.

While the data-driven clusters can represent appropriate subgoals, we combined some of them to create a shorter list of higher-level subgoals similar to the programming task rubric. Column 2 in Table 1 show the combined subgoals. It is worth noting that we also took the insights of two instructors of the CS0 course on how meaningful these subgoals are for students to understand. Additionally, because

one of our goals is to use these data-driven subgoals in learning environments, we asked human experts to label them to make them easily understandable by students, and instructors. For example, the human label of subgoal 1 is: "Create a procedure with one parameter, and use it in your code", as shown in Table 1. We then developed a data-driven subgoal detector that takes student code as an input, and outputs the status of each subgoal. For example, if the input is code C and the output is `{1, 0, 0, 1}`, this means the subgoal detector detects the completion of subgoals 1 and 4, and the absence of subgoals 2 and 3 in C.

3.2.2 Step 2: Investigating Data-Driven Subgoals

The goal of this step is to investigate the correctness of the *DD* subgoal detections. In Spring 2020 (S20), we integrated the *DD* subgoal detector into the Snap*!* programming environment for the *Squiral* exercise to provide students with subgoal feedback [39]. In Snap*!*, students could see the subgoal labels (shown in Table 1), colored gray to start, green when the subgoal was detected, or red when it became broken. We collected 4,480 edit logs from 27 student submissions with an average of 166 edits per student in S20. For each student edit, we recorded the *DD* subgoal detection state (e.g. `{1, 0, 0, 0}` for subgoal 1 being complete). We asked human experts to manually replay each student's trace data and evaluate whether the subgoal labels, as shown to students, were achieved at the specific timestamps when the *DD* algorithm detected them. Importantly, we asked experts to report when the expert-authored *labels* for each subgoal (which students saw) were achieved. Since these labels do not *precisely* match the code shape combinations that the *DD* subgoal detector used, it was very possible for the *DD* model to be "wrong." In other words, we asked experts to determine when each student achieved "Create a *Squiral* procedure with one parameter and use it in your code," and compared that to when the *DD* detector marked this subgoal label as complete.

We classified each evaluated instance as either `Early`, `onTime`, or `Late`. An instance is classified as `Early` if the *DD* detection is *before* the human expert detection timestamp, `OnTime` if they coincide, and otherwise `Late`. For example, if for student S_j, the human expert detected the completion of subgoal `i` (SG_i) at time `t` $= 5$; while the algorithm detected it at `t` $= 2$, then we label SG_i for S_j as `Early` detection. Then we sorted students in descending order based on the percentage of false detections in their log data, and we took the first 66% of this data ($\sim$ 18 students as a data sample) to investigate the reasons for false detections. We did not use the full set of false detections, since our primary goal was to fix the most common mismatches, without overfitting to the dataset.

We then focus on false detections that occured due to new, correct solutions, in the S20 dataset, that had no matching code shapes in the training dataset (S16, F16, and S17 datasets). We do not investigate expected false detections that occured due to known limitations in the *DD* algorithm (e.g. the *DD* algorithm does not differentiate between variable names).

We found three reasons for false detections for subgoals 1, 2, and 4. Inspired by the design of the constraint-based SqlTutor by Mitrovic et al. [41], we introduce 3 fixes (or constraints) to resolve them.

Subgoal 1 false detection. As shown in Table 1, subgoal 1 label requires a student to create a procedure with one parameter, and use it (or evaluate it) in the main script. However, subgoal 1 code shapes consist of the creation and evaluation of a procedure, ***or*** the use of a `'ReceiveGo'` block (the hat block used to start a script). This means that whether a student created and evaluated a procedure, *or* added a `'ReceiveGo'` block in the main script, the *DD* model will detect the completion of that subgoal, but experts did not interpret the 'ReceiveGo' block as meeting this subgoal, yielding a false detection. To fix this false detection, we simply removed the 'ReceiveGo' block as an option for this subgoal. Code shape A in Figure 2 shows the code shapes of subgoal 1 of the *DD* model (on the left), and how it is fixed in the hybrid model (on the right).

Subgoal 2 false detection. As shown in line 6 in Figure 1, subgoal 2 requires a student to use a 'repeat block that iterates 4 times the number of rotations to draw a *Squiral* with the correct number of sides. While code shapes of this subgoal satisfy this definition, they also include a code shape of adding a 'pen down' block, which is necessary to draw, but *only* inside a procedure. Therefore, if a 'pen down' block is used outside of a procedure, subgoal 2 will not be detected. To fix this false detection, we added a disjunction code shape to detect the presence of `'pen down'` inside *or* outside a procedure, as shown in code shape B in Figure 2.

Subgoal 4 false detection. As shown in lines 6-9 in Figure 1, subgoal 4 requires the use of 'move', 'turn', and 'change - by -' blocks (which increments a value of a variable), inside a 'repeat block'. We found that code shapes of subgoal 4 only include the `'turnLeft'` block; however, if the student solution contains a `'turnRight'` block (which does the same 'turn' functionality but from a different direction), the subgoal will not be detected. To fix this false detection, we modified all the code shapes that require the use of `'turnLeft'` block to accept either `'turnRight'` or `'turnLeft'` blocks, as shown in code shape C in Figure 2.

These three false detections show that prior solutions in S16, F16, and S17 datasets often used a `'ReceiveGo'` and `'turnLeft'` blocks, and used `'pen down'` inside a procedure; but this was not always the case in the S20 data. This shows that investigating the accuracy of a model, either data-driven or expert, is necessary since it is impossible to predict how students will behave in practice or how the data will change from one class to another.

3.2.3 Step 3: Improving the Data-Driven Subgoals with Human Insights

The goal of this step is to apply the human expert constraints (explained in Step 2) to mitigate the false detections of the *DD* algorithm. To do so, we developed a tool that iterates over each code shape of the data-driven subgoals, and allows humans to edit code shapes (i.e. add, delete or modify) to apply the three constraints (i.e. fixes) explained in Step 2. Because this tool modifies the code shapes, we then use the original *DD* algorithm to re-cluster all code shapes to ensure that the most similar code shapes remain

Table 1: Data-driven subgoals (composed of code shape clusters) with their corresponding human labels that were used when presented to students.

Data-Driven Code Shape Clusters	Combined Clusters	Subgoal Human Label
C1: Evaluate a procedure with one parameter on the script area OR Add a 'ReceiveGo' block.	C1 + C2 = Subgoal 1	Create a Squiral procedure with one parameter and use it in your code.
C2: Create a procedure with one parameter.		
C3: Have a 'multiply' block with a variable in a 'repeat' block OR two nested 'repeat' blocks.	C3 + C4 = Subgoal 2	The Squiral procedure rotates the correct number of times.
C4: Have a 'pen down' block inside a procedure		
C5: Add a variable in a 'move' block inside a 'repeat' block.	C5 = Subgoal 3	The length of each side of the Squiral is based on a variable.
C6: Have a 'move' and 'turnLeft' block inside a 'repeat' block.	C6 + C7 = Subgoal 4	The length of the Squiral increases with each side.
C7: 'Change' a variable value inside a 'repeat' block.		

clustered together. Figure 2 shows an example of three code shapes before and after they have been edited by human experts, that fix false detections that existed for subgoals 1, 2, and 4, respectively.

In summary, our hybrid model **humanizes** data-driven subgoal detection models through a series of important steps. First, we apply a data-driven model to correct, historical student solutions to generate a set of human *interpretable* code shape clusters. Second, a human expert labels the subgoals these clusters represent, in a way that is meant to align with the original programming assignment. Third, we collect data from students solving the same task using a programming environment augmented with subgoal feedback (i.e human labels with colors) based on the *DD* subgoal detector. Fourth, we had experts examine code traces with the *DD* subgoal feedback to determine when the displayed subgoal labels were actually achieved. Fifth, human experts modified the code shapes that led to discrepancies between the data-driven and expert detections for the displayed subgoals. This series of steps leverages the natural cycle of a frequently-offered CS0 class to bootstrap the creation of *DD* subgoal detectors in the programming environment.

4. EVALUATION

In this experiment, we applied both the hybrid and the *DD* models to detect subgoals in students' S20 code submissions. We also asked two human experts to evaluate the presence or absence of each subgoal in a subset of students' code snapshots (sequential states of student code, corresponding to their code edits, e.g., the addition or deletion of code blocks) using the subgoal labels (shown in Table 1) as rubric items (with 1 for the subgoal's presence and 0 for its absence), resulting in an *expert (or gold standard) subgoal state*.

Because S20 data consists of 4480 code snapshots, it is not feasible to evaluate the models on every timestamp for two reasons. First, students mostly need feedback on a given subgoal when they are making edits towards finishing *that subgoal*, not after every single edit they make. Second, students break and recomplete subgoals frequently, even when they are not working on a particular subgoal, and therefore it is not meaningful to have an expert label at every single datapoint. As a result, we evaluate the models at the most meaningful times when a student is close to finishing a subgoal, including: (1) the first time a student completed a subgoal, according to a human expert, (2) up to five code edits before that subgoal is completed, and (3) any time when either model (i.e. hybrid, or *DD*) suggests a *change* in a subgoal's status. While these changes may or may not be true, we wanted to have experts evaluate the correctness of how the algorithms may have detected subgoal changes at these points.

For each subgoal, two human experts evaluated 150, 163, 178, and 196 student snapshots for subgoals 1, 2, 3, and 4, respectively, making a total of 687 labeled snapshot datapoints. The experts used the subgoals as their rubric; and they started the labelling process by evaluating *the first time* a subgoal is detected. To do so, they divided the data (27 students * 4 subgoals = 108 datapoint) into a set of 6 rounds, where the first round consists of 2 students and the remaining 5 rounds consists of 5 students. The two experts collaboratively evaluated the first round together to make sure they have a clear understanding of the rubric subgoals. Then for the next two rounds, they evaluated the logs independently and after each, they met to discuss and resolve conflicts. For these 3 rounds, the two experts had a moderate to substantial agreement with a Cohen's kappa ranging from 0.5 to 0.67. The reason why the kappa values are low is that we considered any disagreement even if it was a difference of *one* timestamp, but it does highlight how subgoal detection can be subjective, which is a challenge for measuring the accuracy of subgoal detection. As a result, for the remaining data, the two experts continued to evaluate it independently and then met to discuss and resolve conflicts to produce relatively objective *gold standard* expert labels. We used these labelled logs as ground truth to compare the accuracy and F1-scores of both the *DD* and the hybrid models. We also calculated the agreement between the expert detections and those generated by the hybrid and *DD* models.

4.1 Supervised Learning Models

We also compared our hybrid humanized model with supervised machine learning models as another form of baseline. The supervised models were trained and tested on the S20 dataset, using the same 687 expert-labeled snapshots described above. We trained separate models to detect each of

the subgoal labels (using training/testing splits discussed below). This allows us to directly compare these supervised methods, which require *labeled* training data, to the *DD* model, which does not, and to our hybrid approach, where the expert uses *some* of the labeled data to improve the model. While we discuss some limitations to this comparison in Section 7, these baselines help contextualize the performance of our subgoal detectors.

The baseline supervised machine learning models we have chosen are two shallow models (SVM [14] and XGBoost [9]) and one deep learning model (code2vec [4]). We used the same edits for predictions as the *DD* and hybrid models, and extracted the term frequency–inverse document frequency (TF-IDF) features from the models, thus a vector representation of an edit is generated, and used for training the two models. We performed grid search cross validation for both models. For SVM, we used a parameter space of linear kernel and Radial Basis Function (RBF) [60] kernel, and searched the regularization parameters from 1 to 10. For XGBoost, we searched the subsampling space of 0.1 to 1, with the number of estimators from 5 to 100, stepping by 5. Ten-fold cross validation is performed to search the parameter spaces. The training, validation, and test sets are split by students to make sure that no students used for testing will have an edit in training, since edits from one student would be very similar, and using samples similar to the testing set in training would lead to an unfair comparison.

We selected one state-of-the-art deep learning model, code2vec [4], for comparison as well, as the model has recently been applied in educational code classification tasks [53]. Instead of using a vector of term frequency to represent edits, code2vec uses the structural representations from ASTs to represent the code, and the representation is learned from training a neural network[1]. We used early stopping to avoid overfitting. To ensure the robustness of our results, we ran 20 times with resampling for all supervised baseline models, and reported average metrics (e.g. F1-score, accuracy).

5. RESULTS

RQ1a: *How well does a hybrid model perform compared to a data-driven model without expert augmentation?*

The prediction results for each subgoal from the *DD* model and the hybrid model are shown in Table 2. Our hybrid model achieves higher accuracy and F1-scores on all subgoals than the *DD* model. In particular, it reaches > 0.8 accuracy for all subgoals, and it reaches > 0.9 accuracy for 2 out of the 3 subgoals that were modified (i.e. subgoal 1 and 4) with expert constraints. It is worth noting that the hybrid model achieves higher accuracy in subgoal 3, which was not modified with expert constraints. This is possible because after we modified code shapes for the *other* subgoals, we reclustered the code shapes (as described in Section 3.2.3), and the new code shapes for subgoal 3 were changed. This is likely because, after reclustering, some code shapes moved to subgoal 3, resulting in higher recall.

We also measured the agreement between human experts, *DD*, and hybrid model subgoal detections. For the four subgoals, we find low to moderate agreement over all student logs between *DD* and human expert detections, with Cohen's kappa values ranging between 0.25-0.581. However, we find substantial or better agreement between the hybrid model and human expert detections, with Cohen's kappa values ranging between 0.6-0.84. It is worth noting that this agreement is higher than that achieved between the two human experts (described in Section 4). These results showed that the addition of just three human constraints to a data-driven model succeeded in improving its accuracy, making the hybrid model agree *more* with the gold standard (that the experts co-constructed), than the experts' original agreement with one another.

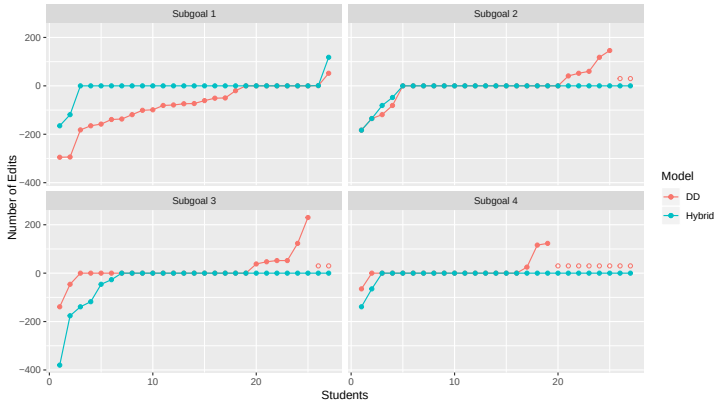

Figure 3: The number of code edits (y-axis) that occurred between gold standard expert subgoal detections, and detections by the DD and Hybrid models, for first-time subgoal detections for each student trace (x-axis).

We also determined the number of false detections (i.e. `Early` and `Late` detections, as described in Section 3.2.2) for both the hybrid and *DD* models. We found the *DD* model detected 40.66%, 10.66%, 5.62% and 5.10% `Early` detections, and 2%, 13.5%, 12.36%, and 15.31% `Late` detections for subgoal 1, 2, 3, and 4, respectively. However, our hybrid model detected 1.33%, 13.5%, 10.67% and 4.6% `Early` detections, and 8%, 5.52%, 1.7%, and 2.81% `Late` detections for subgoal 1, 2, 3, and 4, respectively. To visualize these false detections, Figure 3 visualized these false detections by presenting the distance (i.e. how many edits) between the `Early` and `Late` detections by both the *DD* and hybrid models, and the gold standard human expert detections of the *first time* a subgoal is completed. The x-axis presents the students ($n = 27$), and the y-axis presents the number of edits a student makes until they complete a subgoal. We used a negative number to indicate how much *earlier* the models were than the gold standard detection, 0 to show when models *agree* with the gold standard, and a positive number to show how much *later* the models were. We also used *empty* circles to indicate instances where a subgoal is *never* detected by the models but it was detected by human experts. While Figure 3 shows *only* how early/late the model is in detecting when a subgoal is *first* completed, this is likely the most important detection. Our results suggest a high agreement between the hybrid model's detections with the gold standard, and a strong improvement over the *DD* model.

[1]We applied code2vec using the process described in [53].

Table 2: Precision, Recall, F1-score and Accuracy observed with Supervised, Data-Driven (DD) and our Hybrid Models.

	Precision					Recall					F1-score					Accuracy				
	SVM	XG-Boost	Code2vec	DD	Hybrid	SVM	XG-Boost	Code2vec	DD	Hybrid	SVM	XG-Boost	Code2vec	DD	Hybrid	SVM	XG-Boost	Code2vec	DD	Hybrid
Subgoal 1 (n = 150)	0.71	0.68	0.89	0.44	**0.95**	0.79	0.75	0.91	**0.94**	0.76	0.71	0.66	**0.89**	0.60	0.85	0.82	0.76	0.90	0.57	**0.91**
Subgoal 2 (n = 163)	0.58	0.61	0.57	**0.70**	0.69	0.61	0.77	0.79	0.63	**0.85**	0.55	0.66	0.64	0.66	**0.76**	0.74	0.75	0.75	0.77	**0.81**
Subgoal 3 (n = 178)	0.60	0.62	0.69	**0.80**	0.75	0.54	0.61	0.76	0.64	**0.95**	0.52	0.59	0.70	0.71	**0.84**	0.70	0.72	0.79	0.82	**0.88**
Subgoal 4 (n = 196)	0.62	0.64	0.64	0.79	**0.87**	0.64	0.75	0.84	0.55	**0.93**	0.59	0.66	0.69	0.65	**0.90**	0.76	0.74	0.75	0.80	**0.93**

RQ1b: *How does a hybrid model perform compared to supervised learning approaches that leverage expert subgoal labels?*

We show our comparison of supervised learning models and the hybrid model in Table 2. On all subgoals, except one, the hybrid model has higher accuracy and F1-score than code2vec, SVM, and XGBoost models, outperforming them by 0.10, 0.06 and 0.15 percent of F1-score, respectively. In subgoal 1, we found that code2vec achieved a higher F1-score than all the other models, and a relatively similar accuracy to the hybrid model (0.903, 0.906). One possible explanation for this is that subgoal 1 is the simplest subgoal, requiring only that the student has defined and used a procedure, regardless of its content, and this simple code pattern may have been easier for the supervised approaches to learn. These results show that a hybrid model iteratively constructed through cycles of student data collection, machine learning, along with human labeling and correction can be used to create accurate automatic subgoal detections on a novice programming task. Furthermore, these supervised learning models, that were mostly outperformed by our hybrid model, were learned using labels from snapshots that were *strategically chosen* to reflect important decision points for the model, suggesting that the supervised models' performance may suffer if a random selection of snapshots were used to create an expert-labeled training set instead.

5.1 Case Studies

In this section, we present case studies to highlight ways the hybrid model improved upon the original *DD* model, as well as the hybrid model's limitations. These case studies come from the 33% of students who were *not investigated* when the expert identified false detections in S20 from the original *DD* model, as discussed in our methods (Section 3.2.2). These students also used the original *DD* system, but their data did *not* inform our hybrid model. These case studies, therefore, help us understand the ways our hybrid model might help new students, as well as limitations of the model. Though our prior work suggests the *DD* subgoal detections overall were often useful to students [39], our post-hoc analysis here shows that the *false detections* may have negatively impacted student programming behavior, suggesting the need for our hybrid model's improvements.

5.1.1 Case Study 1 (Em): Inaccurate Data-Driven Subgoal Feedback

We present here a case study of the student Em[2] when they received an inaccurate subgoal detection based on the *DD* model, and how the hybrid model could have mitigated this false detection.

Em started solving *Squiral* by snapping the 'when green flag clicked block' (i.e. 'ReceiveGo' block) on the main script as shown in Figure 4A, and the system *falsely* detected subgoal 1. Em then proceeded to work on subgoal 2, *without* creating the required procedure, and created a loop using the 'repeat' block nested with 'move' and 'turnLeft' blocks (shown in Figure 4B). This time, the system was correct in not detecting subgoal 2 because the loop was not in a procedure, and does not iterate on the 'rotations' parameter. Afterwards, Em correctly created a procedure with one parameter as shown in Figure 4C; however, the system shows no change, since it already falsely detected subgoal 1 earlier, and therefore, no change in the feedback is given to the student. Em then destroyed the procedure, without ever making it again. Em kept working for the rest of the time on creating a number of redundant loops, similar to the one in Figure 4C, with constant values to manually draw the *Squiral* shape (rather than using a variable to vary its length).

Em spent a total of 55 minutes to draw *Squiral* in an iterative manner. While the *DD* system accurately detected subgoals 2-4 as incomplete, this case study highlights potential harm that may have arisen from the false detection of subgoal 1. When subgoal 1 was detected early, Em skipped over creating a procedure. Later, when she did create the procedure correctly, she got no additional feedback (since the subgoal was already detected), and promptly deleted it. Preventing these unneeded deletions is a primary role of correct, positive subgoal feedback. However, had Em been using the hybrid model, subgoal 1 would not have been detected early because the expert edited the faulty code shape. We argue that this might have allowed Em to keep working on creating a procedure (as shown in snapshot C in Figure 4), which would have been detected as complete by the hybrid system only at this time. It is also possible that receiving inaccurate feedback at the very beginning may have led to Em's mistrust in the system, since prior work shows that incorrect feedback can reduce students' willingness to use it [48].

Note that, we do not believe this incorrect *DD* detection

[2]We provide anonymous names for students.

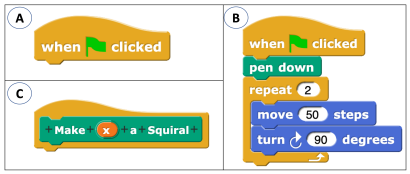

Figure 4: Code snapshots A, B, and C, implemented by student Em in Case Study 1.

means the full data-driven system should not be used since our prior work shows the system can be helpful to students [39]. However, we need to explore how to present subgoal feedback in a way that promotes students to question the feedback since any such program will inevitably fail to recognize some correct variants of a subgoal solution.

5.1.2 *Case Study 2 (Jo): Cases when Hybrid Subgoal Detections could be Incorrect*

We present here a case study where the hybrid model would *not* have provided accurate subgoal feedback. As evidenced by the model's high overall accuracy (Table 2), these instances were rare, but understanding them highlights the affordances and limitations of our approach. Specifically, we investigated subgoal 1, where the hybrid model had the lowest F1-score.

Student Jo correctly created a procedure with a parameter; however, since Jo had not yet used the procedure in their main script, subgoal 1 was not detected (accurate detection). Jo continued programming and completed subgoals 2 and 3, which were accurately detected by the system. Afterwards, Jo added a procedure call to the main script, and subgoal 1 was detected as complete (accurate detection). However, Jo then added a 'pen up' block underneath the procedure call, where unexpectedly, the hybrid model changed subgoal 1's status back to incomplete (incorrect detection).

This false detection in the hybrid model was due to an overly-specific code shape. Specifically, the code shape required the procedure call to be the *last block* in the main script (which was true for 94% of students, but not Jo), leading to the false detection. This case confirms the importance of iteratively investigating and refining data-driven subgoal detections to keep improving their accuracy, which is a common process in expert-authored models as well. While this false detection has a straightforward fix, similar to the ones presented in Section 3.2.2, it shows one limitation of the hybrid model: creating these fixes requires the expert to find and address the false detections in the first place, which is dependent on finding the bugs in the data inspected. This is also one reason why the hybrid model performance of subgoal 1 has a lower F1-score than the code2vec model (as shown in Table 2).

6. DISCUSSION

6.1 Automated Subgoal Detection

The key contribution of this paper is tackling the critical challenge of *automated subgoal detection* during programming tasks. Our results show that a hybrid data-driven model meaningfully addresses this goal, with high accuracy and F1 score when detecting subgoals at key moments during students' work. Our results show that this is a challenging task: even a state-of-the-art supervised learning approach with access to labeled data struggled to identify some subgoals (F1 score as low as 0.64). This agrees with prior work using expert-authored [13, 38] and supervised learning models [53], showing that immediate feedback on subgoals is a hard problem.

While automatically detecting subgoals is challenging, research suggests that the ability to provide automated, immediate feedback on subgoals can significantly improve students' motivation and learning. Providing subgoals for novices can improve student learning by breaking down the programming task into smaller subtasks, which is a challenging task for novices [36, 38]. In human tutoring dialogues for programming, tutors provide a combination of corrective and positive feedback, increasing students' motivation and confidence in programming [8, 33, 17]. Automatic subgoal detection could be used to provide similar corrective and positive feedback during programming. We know of only 3 systems that can afford such immediate feedback, that is not based on unit tests, during programming, that have been shown to promote learning, confidence and persistence for linked lists [21], database queries [42], and block-based programming [38]. It is perhaps uncommon to make such systems due to the difficulty in anticipating all student approaches, paired with the high potential for inaccuracies and student reactions to them. Our accuracy results suggest that our hybrid, humanized approach can be used to build similar automatic subgoal detection systems that could be deployed and more easily scaled across problems in real classrooms.

6.2 Affordances of Data-driven, Hybrid and Expert models

Our results suggest that the hybrid model has good potential for solving the problem of automated subgoal detection. Here we discuss the advantages and trade-offs of the hybrid approach, compared to data-driven and expert models.

6.2.1 *RQ1.a: Hybrid versus Data-Driven Models*

Our results show that a hybrid, iterative model that leverages data-driven subgoal extraction, human labeling, and expert refinement based on labeled student data, can *greatly* improve model performance compared to a purely data-driven (DD) model. The expert constraints improved F1-score of the data-driven model by 0.14-0.25 points, as shown in Table 2. Based on our analysis, the hybrid model, reduced the number of `Early` and `Late` subgoal detections and increased the `onTime` detections, when compared to the original *DD* model, as shown in Figure 3. This is a critical improvement, since prior work shows that the quality of feedback affects novices' programming behavior [48, 51], but also their self-perceptions, and trust in the learning environment [48].

The hybrid model creation does require additional labelling effort needed to evaluate the models; however, this effort seems well worth it, and is needed to evaluate the accuracy of any data-driven model before deployment. Compared to prior work, our iterative hybrid model shares similar benefits of "human-in-the-loop" methods in machine learning [56, 30] and also represents data-driven rules in an interpretable and

editable form that simplified the process of merging human insights into the model. In prior work, Diana et al. found, qualitatively, that their generated data-driven rubrics are considerably similar to human-generated rubrics [19]. Similarly, in our work, not only did instructors agree with the hybrid model subgoal detections, we also found these detections have substantial or better agreement with the human expert gold standard than the *DD* model. From these results, we conclude that **a hybrid model can be used to iteratively improve and humanize data-driven subgoal detection.**

6.2.2 RQ1b: Hybrid versus Supervised Learning Models that Leverage Expert Subgoal Labels

Our results show that **a hybrid model can surpass the performance of supervised learning models.** The steps we used to create our hybrid model were to: (1) apply a data-driven model, (2) add expert constraints, and (3) determine interesting datapoints consisting of times when subgoals might be achieved. The steps we used to create supervised learning models leveraged the interesting datapoints from step 3 of our hybrid model, hand-labeled them, and used them to build supervised learning models. We highlight this to point out that it is hard to determine what labeled data to use in a supervised learning model for subgoal detection, since there are hundreds of potential snapshots from each student. As a result, it is unclear whether the supervised model would have performed as well, compared to one learned from a labeled dataset selected at random or regular timestamps. Our results show that, even after using carefully selected labelled data to train the model, the SVM and XGBoost baselines did not achieve the level of accuracy of the hybrid model for all subgoals. Even code2vec, the state-of-the-art supervised learning model [53], has lower performance for all subgoals, except subgoal 1, than the hybrid model. Perhaps the code2vec performed worse due to the size of labelled dataset (687 datapoints), though recent results suggest the model is still effective with small datasets [52].

6.2.3 Hybrid versus Expert Models

Our hybrid approach offers distinct advantages and trade-offs compared to expert-authored models for subgoal detection. Traditional expert-authored models (e.g. constraint-based tutors [42]) have the advantage of high accuracy and high confidence, but the trade-offs of considerable domain expert time for creation and the potential failure to anticipate some student strategies and misconceptions. Our hybrid model has the advantage of incorporating actual student strategies and misconceptions, and primarily requires human effort to *label data* and identify errors, tasks which can potentially be done by non-experts and distributed across multiple people. A domain expert is only needed to edit the automatically extracted rules and is afforded the chance to do so with actual student data available. A significant tradeoff of the hybrid model is its reliance on data - so the quality of the dataset will directly impact the quality of the subgoal detectors. Furthermore, both models are likely to need refinement as students use them, and this process is already built into the hybrid model creation and refinement cycle.

7. LIMITATIONS & CONCLUSION

This work has 5 main limitations. First, while the *DD* model can capture small differences in solution approaches in *Squiral* (like having whether a 'turnLeft' or 'turnRight' block), we have not tested it in programming tasks with larger space of solution approaches. Therefore, it is not known how well the accuracy results will generalize to other types of programming tasks or languages. However, the iterative process of data collection, *DD* subgoal extraction, labeling, and collection of data from students using the subgoal labels and detectors, could be applied for other programming problems, of the same level as *Squiral*, and repeated until the models achieved high accuracy. Second, some of S20 data that was used for models' evaluation was also used to inform expert constraints in the hybrid model. However, this was only 66% of the data, and we discussed above how the added constraints are generalizable, which should have helped in any semester (see Section 5.1). Additionally, our case studies in Section 5.1 show examples of how the hybrid detector performed on unseen data, though there was insufficient data for a quantitative evaluation.

Third, we used only the labelled S20 data to train the supervised baseline models, but we also used 3 other semesters of *unlabeled* data to train the unsupervised *DD* and hybrid models. However, we argue that this ability to leverage a larger unlabeled dataset is an advantage of the unsupervised methods, rather than a limitation of our analysis. Fourth, some of the datapoints that were labeled for the evaluation of all the models were selected in part by using the hybrid and *DD* model detections, as discussed above, and this might have biased the results. However, all the models were evaluated on these same datapoints that were *strategically* chosen for their importance, and there are instances where some of the supervised models outperformed the original *DD* model. It is not clear how a different data selection strategy would have affected the results, and we argue that training and testing the supervised models on a dataset of the same size with randomly selected snapshots would likely decrease the performance of supervised models. Finally, we did not compare the hybrid model to a purely expert-authored model, and we did not measure time taken by experts to modify the data-driven rules. We argue that these comparisons require hiring experts to author rules and performing time analysis, which is beyond the scope of this paper.

In summary, this work proposes a new paradigm for 'humanizing' data-driven subgoal detection for novice programming. Specifically, we proposed to humanize data-driven subgoals in an iterative refinement process. We (1) extract data-driven subgoals from student work, (2) give them human labels, (3) collect more data from students programming with the labels and subgoal detectors, (4) present experts with the labels, and interpretable detectors, along with student behavior data so they can add expert constraints. This process ensures that humans are involved in every step of the creation of automatic subgoals, offering the advantages of reflecting real student behaviors, and limiting and focusing expert authoring time. Our results show that this hybrid humanized model outperforms fully data-driven models and state-of-the-art supervised learning models. This proposed paradigm can be used to create humanized automatic subgoal detection for tasks where it may be too expensive to create full expert models for, but that are important for student learning, motivation, and retention.

8. REFERENCES

[1] V. Aleven, B. M. McLaren, and J. Sewall. Scaling up programming by demonstration for intelligent tutoring systems development: An open-access web site for middle school mathematics learning. *IEEE transactions on learning technologies*, 2(2):64–78, 2009.

[2] V. Aleven, B. M. McLaren, J. Sewall, M. Van Velsen, O. Popescu, S. Demi, M. Ringenberg, and K. R. Koedinger. Example-tracing tutors: Intelligent tutor development for non-programmers. *International Journal of Artificial Intelligence in Education*, 26(1):224–269, 2016.

[3] U. Alon, M. Zilberstein, O. Levy, and E. Yahav. A general path-based representation for predicting program properties. *PLDI'18*, 2018.

[4] U. Alon, M. Zilberstein, O. Levy, and E. Yahav. code2vec: Learning distributed representations of code. *POPL'19*, 2019.

[5] J. R. Anderson, A. T. Corbett, K. R. Koedinger, and R. Pelletier. Cognitive tutors: Lessons learned. *The journal of the learning sciences*, 4(2):167–207, 1995.

[6] D. Azcona, P. Arora, I.-H. Hsiao, and A. Smeaton. user2code2vec: Embeddings for profiling students based on distributional representations of source code. In *LAK'19*, 2019.

[7] M. Ball. Lambda: An Autograder for *snap*. Technical report, Electrical Engineering and Computer Sciences University of California at Berkeley, 2018.

[8] K. E. Boyer, R. Phillips, M. D. Wallis, M. A. Vouk, and J. C. Lester. Learner characteristics and feedback in tutorial dialogue. In *Proceedings of the Third Workshop on Innovative Use of NLP for Building Educational Applications*, pages 53–61. Association for Computational Linguistics, 2008.

[9] T. Chen, T. He, M. Benesty, V. Khotilovich, Y. Tang, H. Cho, et al. Xgboost: extreme gradient boosting. *R package version 0.4-2*, 1(4), 2015.

[10] C.-Y. Chou, B.-H. Huang, and C.-J. Lin. Complementary machine intelligence and human intelligence in virtual teaching assistant for tutoring program tracing. *Computers & Education*, 57(4):2303–2312, 2011.

[11] A. Corbett and J. R. Anderson. Locus of Feedback Control in Computer-Based Tutoring: Impact on Learning Rate, Achievement and Attitudes. In *Proceedings of the SIGCHI Conference on Human Computer Interaction*, pages 245–252, 2001.

[12] A. Corbett, L. Kauffman, B. Maclaren, A. Wagner, and E. Jones. A cognitive tutor for genetics problem solving: Learning gains and student modeling. *Journal of Educational Computing Research*, 42(2):219–239, 2010.

[13] A. T. Corbett and J. R Anderson. Knowledge decomposition and subgoal reification in the act programming tutor. 1995.

[14] C. Cortes and V. Vapnik. Support-vector networks. *Machine learning*, 20(3):273–297, 1995.

[15] S. Custer, E. M. King, T. M. Atinc, L. Read, and T. Sethi. Toward data-driven education systems: Insights into using information to measure results and manage change. *Center for Universal Education at The Brookings Institution*, 2018.

[16] J. Denner and L. Werner. Computer programming in middle school: How pairs respond to challenges. *Journal of Educational Computing Research*, 37(2):131–150, 2007.

[17] B. Di Eugenio, D. Fossati, S. Ohlsson, and D. Cosejo. Towards explaining effective tutorial dialogues. In *Annual Meeting of the Cognitive Science Society*, pages 1430–1435, 2009.

[18] N. Diana, M. Eagle, J. Stamper, S. Grover, M. Bienkowski, and S. Basu. Data-driven generation of rubric parameters from an educational programming environment. In *International Conference on Artificial Intelligence in Education*, pages 490–493. Springer, 2017.

[19] N. Diana, M. Eagle, J. Stamper, S. Grover, M. Bienkowski, and S. Basu. Data-driven generation of rubric criteria from an educational programming environment. In *Proceedings of the 8th International Conference on Learning Analytics and Knowledge*, pages 16–20, 2018.

[20] M. L. Epstein, A. D. Lazarus, T. B. Calvano, K. A. Matthews, R. A. Hendel, B. B. Epstein, and G. M. Brosvic. Immediate feedback assessment technique promotes learning and corrects inaccurate first responses. *The Psychological Record*, 52(2):187–201, 2002.

[21] D. Fossati, B. Di Eugenio, S. Ohlsson, C. Brown, and L. Chen. Data driven automatic feedback generation in the ilist intelligent tutoring system. *Technology, Instruction, Cognition and Learning*, 10(1):5–26, 2015.

[22] D. Garcia, B. Harvey, and T. Barnes. The beauty and joy of computing. *ACM Inroads*, 6(4):71–79, 2015.

[23] V. G. Goecks. Human-in-the-loop methods for data-driven and reinforcement learning systems. *arXiv preprint arXiv:2008.13221*, 2020.

[24] L. Gusukuma, A. C. Bart, D. Kafura, and J. Ernst. Misconception-driven feedback: Results from an experimental study. In *Proceedings of the 2018 ACM Conference on International Computing Education Research*, pages 160–168, 2018.

[25] L. Gusukuma, D. Kafura, and A. C. Bart. Authoring feedback for novice programmers in a block-based language. In *2017 IEEE Blocks and Beyond Workshop (B&B)*, pages 37–40. IEEE, 2017.

[26] P. Ihantola, T. Ahoniemi, V. Karavirta, and O. Seppälä. Review of recent systems for automatic assessment of programming assignments. In *Proceedings of the 10th Koli calling international conference on computing education research*, pages 86–93, New York, NY, 2010. ACM.

[27] J. Jeuring, L. T. van Binsbergen, A. Gerdes, and B. Heeren. Model solutions and properties for diagnosing student programs in ask-elle. In *Proceedings of the Computer Science Education Research Conference*, pages 31–40, 2014.

[28] D. E. Johnson. Itch: Individual testing of computer homework for scratch assignments. In *Proceedings of the 47th ACM Technical Symposium on Computing Science Education*, pages 223–227. ACM, 2016.

[29] D. E. Johnson. Itch: Individual testing of computer homework for scratch assignments. In *Proceedings of the 47th ACM Technical Symposium on Computing*

Science Education, pages 223–227, New York, NY, 2016. ACM.

[30] B. Kim. *Interactive and interpretable machine learning models for human machine collaboration*. PhD thesis, Massachusetts Institute of Technology, 2015.

[31] N. Körber, K. Geldreich, A. Stahlbauer, and G. Fraser. Finding anomalies in scratch assignments. *arXiv preprint arXiv:2102.07446*, 2021.

[32] E. Lahtinen, K. Ala-Mutka, and H.-M. Järvinen. A study of the difficulties of novice programmers. *Acm Sigcse Bulletin*, 37(3):14–18, 2005.

[33] M. R. Lepper, M. Woolverton, D. L. Mumme, and J. Gurtner. Motivational techniques of expert human tutors: Lessons for the design of computer-based tutors. *Computers as cognitive tools*, 1993:75–105, 1993.

[34] A. Luxton-Reilly, I. Albluwi, B. A. Becker, M. Giannakos, A. N. Kumar, L. Ott, J. Paterson, M. J. Scott, J. Sheard, and C. Szabo. Introductory programming: a systematic literature review. In *Proceedings Companion of the 23rd Annual ACM Conference on Innovation and Technology in Computer Science Education*, pages 55–106, 2018.

[35] L. E. Margulieux and R. Catrambone. Finding the best types of guidance for constructing self-explanations of subgoals in programming. *Journal of the Learning Sciences*, 28(1):108–151, 2019.

[36] L. E. Margulieux, R. Catrambone, and M. Guzdial. Employing subgoals in computer programming education. *Computer Science Education*, 26(1):44–67, 2016.

[37] L. E. Margulieux, M. Guzdial, and R. Catrambone. Subgoal-labeled instructional material improves performance and transfer in learning to develop mobile applications. In *Proceedings of the ninth annual international conference on International computing education research*, pages 71–78, 2012.

[38] S. Marwan, G. Gao, S. Fisk, T. W. Price, and T. Barnes. Adaptive immediate feedback can improve novice programming engagement and intention to persist in computer science. In *Proceedings of the 2020 ACM Conference on International Computing Education Research*, pages 194–203, 2020.

[39] S. Marwan, T. W. Price, M. Chi, and T. Barnes. Immediate data-driven positive feedback increases engagement on programming homework for novices. In *Educational Data Mining in Computer Science Education (CSEDM) Workshop @ EDM'20*, 2020.

[40] J. McKendree. Effective feedback content for tutoring complex skills. *Human-computer interaction*, 5(4):381–413, 1990.

[41] A. Mitrovic and S. Ohlsson. Evaluation of a constraint-based tutor for a database language. 1999.

[42] A. Mitrovic, S. Ohlsson, and D. K. Barrow. The effect of positive feedback in a constraint-based intelligent tutoring system. *Computers & Education*, 60(1):264–272, 2013.

[43] D. Palmer. A motivational view of constructivistâinformed teaching. *International Journal of Science Education*, 27(15):1853–1881, 2005.

[44] G. D. Phye and T. Andre. Delayed retention effect: attention, perseveration, or both? *Contemporary Educational Psychology*, 14(2):173–185, 1989.

[45] T. W. Price, Y. Dong, and D. Lipovac. iSnap: Towards Intelligent Tutoring in Novice Programming Environments. In *Proceedings of the ACM Technical Symposium on Computer Science Education*, 2017.

[46] T. W. Price, Z. Liu, V. Catete, and T. Barnes. Factors Influencing Students' Help-Seeking Behavior while Programming with Human and Computer Tutors. In *Proceedings of the International Computing Education Research Conference*, 2017.

[47] T. W. Price, R. Zhi, and T. Barnes. Evaluation of a Data-driven Feedback Algorithm for Open-ended Programming. In *Proceedings of the International Conference on Educational Data Mining*, 2017.

[48] T. W. Price, R. Zhi, and T. Barnes. Hint Generation Under Uncertainty: The Effect of Hint Quality on Help-Seeking Behavior. In *Proceedings of the International Conference on Artificial Intelligence in Education*, 2017.

[49] K. Rivers and K. R. Koedinger. Data-Driven Hint Generation in Vast Solution Spaces: a Self-Improving Python Programming Tutor. *International Journal of Artificial Intelligence in Education*, 27(1):37–64, 2017.

[50] C. Romero and S. Ventura. Educational data mining and learning analytics: An updated survey. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 10(3):e1355, 2020.

[51] P. Shabrina, S. Marwan, T. W. Price, M. Chi, and T. Barnes. The impact of data-driven positive programming feedback: When it helps, what happens when it goes wrong, and how students respond. In *Educational Data Mining in Computer Science Education (CSEDM) Workshop @ EDM'20*, 2020.

[52] Y. Shi, Y. Mao, T. Barnes, M. Chi, and T. W. Price. More with less: Exploring how to use deep learning effectively through semi-supervised learning for automatic bug detection in student code. *EDM*, 2021.

[53] Y. Shi, K. Shah, W. Wang, S. Marwan, P. Penmetsa, and T. Price. Toward semi-automatic misconception discovery using code embeddings. In *The 11th International Conference on Learning Analytics Knowledge (LAK 21)*, 2021.

[54] V. J. Shute. Focus on formative feedback. *Review of educational research*, 78(1):153–189, 2008.

[55] D. Sleeman, A. E. Kelly, R. Martinak, R. D. Ward, and J. L. Moore. Studies of diagnosis and remediation with high school algebra students. *Cognitive Science*, 13(4):551–568, 1989.

[56] R. Souza, L. Neves, L. Azevedo, R. Luiz, E. Tady, P. R. Cavalin, and M. Mattoso. Towards a human-in-the-loop library for tracking hyperparameter tuning in deep learning development. In *LADaS@ VLDB*, pages 84–87, 2018.

[57] J. Stamper, T. Barnes, and M. Croy. Enhancing the automatic generation of hints with expert seeding. *International Journal of Artificial Intelligence in Education*, 21(1-2):153–167, 2011.

[58] J. Stamper, T. Barnes, L. Lehmann, and M. Croy. The hint factory: Automatic generation of contextualized help for existing computer aided instruction. In *Proceedings of the 9th International Conference on Intelligent Tutoring Systems Young*

Researchers Track, pages 71–78, 2008.

[59] D. Toll, A. Wingkvist, and M. Ericsson. Current state and next steps on automated hints for students learning to code. In *2020 IEEE Frontiers in Education Conference (FIE)*, pages 1–5. IEEE, 2020.

[60] J.-P. Vert, K. Tsuda, and B. Schölkopf. A primer on kernel methods. *Kernel methods in computational biology*, 47:35–70, 2004.

[61] D. Wang, W. Dong, and Y. Zhang. Collective intelligence for smarter neural program synthesis. In *2020 35th IEEE/ACM International Conference on Automated Software Engineering Workshops (ASEW)*, pages 98–104. IEEE, 2020.

[62] L. E. Winslow. Programming pedagogy—a psychological overview. *ACM Sigcse Bulletin*, 28(3):17–22, 1996.

[63] D. Xin, L. Ma, J. Liu, S. Macke, S. Song, and A. Parameswaran. Accelerating human-in-the-loop machine learning: challenges and opportunities. In *Proceedings of the Second Workshop on Data Management for End-To-End Machine Learning*, pages 1–4, 2018.

[64] R. Zhi, T. W. Price, N. Lytle, and T. Barnes. Reducing the State Space of Programming Problems through Data-Driven Feature Detection. In *Proceedings of the Educational Data Mining in Computer Science Education Workshop at the International Conference on Educational Data Mining*, 2018.