The PyramidSnapshot Challenge: Understanding student process from visual output of programs

Lisa Yan
Stanford University
yanlisa@stanford.edu

Nick McKeown
Stanford University
nickm@stanford.edu

Chris Piech
Stanford University
piech@cs.stanford.edu

ABSTRACT

In the ideal CS1 classroom, we should understand programming process—how student code evolves over time. However, for graphics-based programming assignments, the task of understanding and grading final solutions, let alone thousands of intermediate steps, is incredibly labor-intensive. In this work, we present a challenge, a dataset, and a promising first solution to autonomously use image output to identify functional, intermediate stages of a student solution. By using computer vision techniques to associate visual output of intermediate student code with functional progress, we supplement a lot of the teacher labor associated with understanding graphics-based, open-ended assignments. We hope our publication of the dataset used in our case study sparks discussion in the community on how to analyze programs with visual program output.

CCS CONCEPTS

• **Social and professional topics** → **Computing education**; • **Computing methodologies** → **Computer vision**;

KEYWORDS

Programming courses; student process; teaching at scale; undergraduate courses; computer vision

ACM Reference Format:
Lisa Yan, Nick McKeown, and Chris Piech. 2019. The PyramidSnapshot Challenge: Understanding student process from visual output of programs. In *Proceedings of 50th ACM Technical Symposium on Computer Science Education, Minneapolis, MN, USA, February 27–March 2, 2019 (SIGCSE'19),* 7 pages.
https://doi.org/10.1145/3287324.3287386

1 INTRODUCTION

First-time CS students learn by programming—they must design an approach, debug their code, and iteratively improve towards a final solution. For a teacher, however, a single timestamped submission per student at the end of this process is insufficient to capture all the intermediate steps a student has taken towards a solution. Even when such progress data is available, it is often intractable to analyze this data in a meaningful way that maps student code to *milestones*, or incremental attempts towards the assignment goal. If teachers could characterize individual and aggregate progress through an assignment, they could better understand how their students think about course concepts.

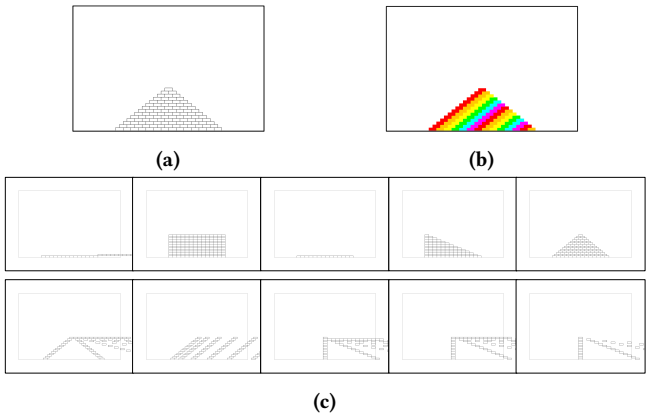


Figure 1: The Pyramid assignment. (a) A full-credit and (b) an extra credit final submission. (c) A series of image outputs of a typical student (top) and a student either tinkering or taking an incorrect path (bottom).

However, student process is highly variable. Unit testing and other automated assessment tools are often only designed to test functionality of the final submission, and teachers would have to design and engineer additional tests for identifying intermediate milestones. Existing tools are also difficult to tailor to *graphics-based* programming assignments, which have recently gained popularity in many CS1 courses. Due to their open-ended nature—enabling programmers of all levels to get quick, visual feedback in an exploratory environment—the large solution spaces often render unit-testing development or syntax-based code analysis insufficient for final submissions, much less intermediate code snapshots.

Despite the up-front complexity of autonomously analyzing functionality of graphics-based coding tasks, advances in computer vision in other fields have matched [14] and, more recently, surpassed [36] human ability to detect objects from pixel input. In spite of their effectiveness, contemporary vision classification techniques have rarely been applied to student code, as many state-of-the-art techniques are *supervised*—requiring well-labeled, plentiful data, which student assignment data often lacks.

In this work, we marry the two fields of understanding students and improving computer vision by presenting the community with a computer vision challenge rooted in CS education: autonomously characterize student progress on a graphics-based programming task by looking at intermediate image output. The PyramidSnapshot dataset[1] is a large, annotated set of images, each tagged with one

SIGCSE'19, February 27–March 2, 2019, Minneapolis, MN, USA

ACM ISBN 978-1-4503-5890-3/19/02…$15.00
https://doi.org/10.1145/3287324.3287386

[1] Dataset available at http://stanford.edu/~cpiech/pyramidsnapshot/challenge.html.

of 16 milestone labels mapping to functional progress towards the assignment goals of Pyramid, a canonical CS1 graphics-based task (Figure 1). The dataset contains timestamped, program image output of 2633 students over 26 CS1 offerings from the same university, corresponding to 101,636 images of intermediate work, of which 84,127 are annotated with milestones.

Our work is a first step towards a deeper understanding of how students work through graphics-based assignments. We hope that our publication of the PyramidSnapshot dataset is a useful case study on how to prepare graphics-based student data for quantitative functional analysis, as well as how to glean pedagogical insights from such fine-grained data. In this work we describe our approach to labeling a complex dataset by hand, and we show that a neural network-based classifier can extend human labeling effort reasonably well. Finally, we share our insights into how to visualize student process to identify different student work patterns.

2 RELATED WORK

There is a growing body of work on automated assessment tools, which are designed to understand and give feedback to students. Many courses also employ test-driven learning, where students use a provided set of unit tests or write their own using a system like Web-CAT [8, 13]. However, unit tests in general are often brittle, time-consuming to develop, and hard to apply to graphics assignments, which allow for variation among correct solutions. Thus, most contemporary work aims to understand student programs based on abstract syntax tree (AST) structure [10, 19, 21, 29, 32]. While these approaches work for short programs with low complexity, Huang et al. found that implementing AST-based feedback in general is as hard a task as autonomously understanding natural language [11].

To make CS courses more accessible for a diverse set of learners, classrooms should promote multiple paths for learning [31]. Some programming languages inherently encourage exploration, such as Scratch [27] and Alice [7]. For text-based programming languages, on the other hand, creativity is pushed to the assignments, which are often open-ended tasks that support visual output [15, 22, 28, 30]. Flexible assignments support not only *planners*, who strategically plan their route to the answer, but also *tinkerers*, who make incremental, exploratory steps towards the solution [3]. Assignment work patterns are often strong indicators of student performance in the course, from debugging behavior [17] and subgoal planning [20] to likelihood for plagiarism [35] and amount of tinkering [5].

It is difficult to autonomously understand the final solution that a student submits. It is even harder to derive insights from the process of how a student solution evolves over time. Existing work that characterizes student work patterns often use low-level indicators of student progress, such as code diffs, completion times, or errors [1, 2, 5, 6, 12]. While there have been studies on functional progress of student solutions in variable-constrained or block-based environments [24, 25, 33], our work is the first to identify program functionality over the course of an open-ended graphics assignment. We circumvent the difficulty of AST-based functionality analysis by using *pixel-based* visual program output.

In 2013, a team of researchers from DeepMind demonstrated that a convolutional neural network based algorithm could learn to play Atari games as well as humans using only pixel output [18]. Because the Atari games used in the DeepMind paper are very similar in complexity to the outputs of assignments typical in CS1 and CS2 classes—and some of the Atari games are in fact classic homework assignments [22]—we have reason to believe that modern computer vision algorithms should be able to understand the output of our student's graphical programs from the pixel level. Despite this potential, to the best of our knowledge, the capacity for understanding graphics prior to this paper has been used mostly to play and rarely to educate. Open datasets have been integral to the early evolution of computer vision techniques [9, 14, 16]; we hope that our dataset will help the CS education community evolve.

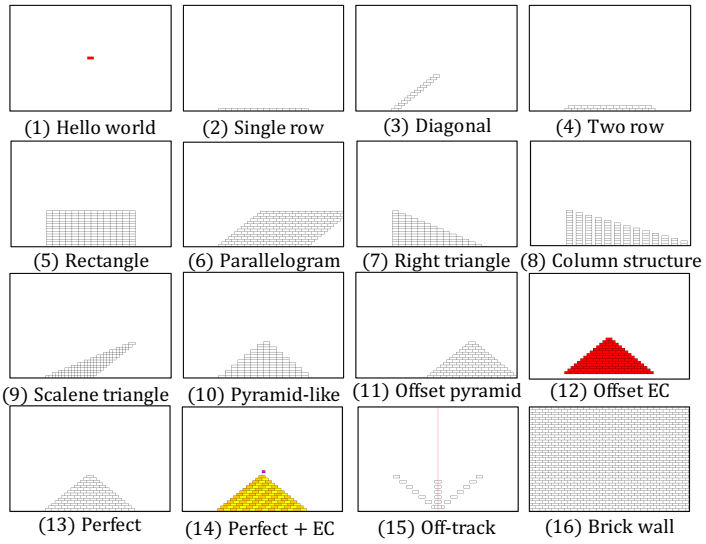
3 DATA

The Pyramid assignment is a sub-problem in the second week of a 10-week CS1 course at Stanford. The assignment is scoped as the students' first exposure to variables, object manipulation, and for-loop indexing. As part of their individual, take-home assignment, students use Java's ACM graphics library to draw bricks on the screen and construct a pyramid shape (Figure 1a), and they are awarded extra credit points if they can extend the correct pyramid (Figure 1b). The solution requires nested loops and the computation of several intermediate variables:

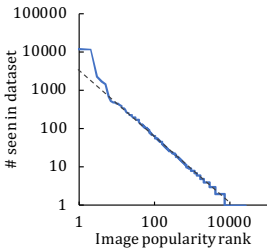
```
public void run()          // draw a pyramid
  for(int i = 0; i < BRICKS_IN_BASE; i++)
    // calculate row variables
    int nBricks = BRICKS_IN_BASE - i;
    int rowWidth = nBricks * BRICK_WIDTH;
    double rowY = HEIGHT-(i+1)*BRICK_HEIGHT;
    double rowX = (WIDTH - rowWidth)/2.0;
    // draw a single row
    for(int j = 0; j < nBricks; j++)
      // add a single brick
      double x = rowX + j * BRICK_WIDTH;
      rect(x, rowY, BRICK_WIDTH, BRICK_HEIGHT);
```

To collect intermediate student data, we modified the Eclipse Integrated Development Environment (IDE) to support *snapshotting* of student code as they work through the assignment. [25]. The IDE manages a git repository that stores a snapshot of student code whenever a student compiles and runs the assignment, meaning that snapshot frequency could be on the order of minutes when a student is actively working. Then, when the student submits their final program for grading, the IDE automatically packages up the repository and stores it on the course server.

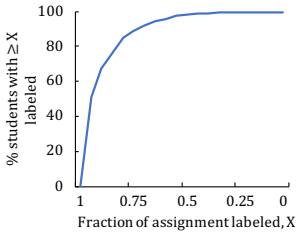
In the CS1 course studied, the Pyramid assignment has been used for many years, with little variation in assignment scope and grading rubric. We analyzed 2,633 student submissions from as far back as 2007 (where the bulk of data was collected between 2012 and 2014) by compiling and running all Pyramid code files in git repositories to generate timestamped images. Figure 3 shows the distributions of the number of snapshots ($\mu = 52, \sigma = 59$) and hours spent ($\mu = 1.7, \sigma = 1.5$) for each pyramid submission. Of the 138,531 snapshots in our dataset, we successfully generated 101,636 images; 36,895 snapshots had runtime or compile errors.



(a)



(b)

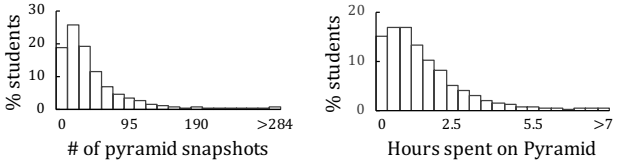


(c)

	# Snapshots	Unique images	Effort score
Total	138531	27220	5.1
Labeled	84127	12077	7.0
Unlabeled	17509	15143	1.2
Error	36895	–	–

(d)

Figure 2: (a) Examples from the 16 milestone category labels in the PyramidSnapshot dataset; EC stands for Extra Credit. (b) Zipf distribution of the dataset images. (c) CDF of the fraction of a repository labeled with the effort strategy in Section 3.2. (d) Label coverage of the PyramidSnapshot dataset.



(a) (b)

Figure 3: Pyramid work time analysis: (a) number of snapshots per student; (b) hours spent on the assignment.

3.1 Dataset complexity

For a graphics-based assignment like Pyramid, we claim that using image output to represent the functionality of student work is less complex—and therefore easier to work with—than using a text-based representation like Abstract Syntax Trees (ASTs). Like many other coding assignment datasets that have been analyzed in the past, the frequency distributions of both the Pyramid assignment code Abstract Syntax Trees (ASTs) and the image output files follow Zipf's law, the same distribution as natural language. One of the implications of this insight is that the exponent s of the Zipf fit can be used as a measure of the complexity of a programming dataset, where a higher exponent means that a dataset has a higher probability of observing the same solution more than once and is thus less complex [23]. The frequency distribution of Pyramid ASTs fits to a Zipf exponent $s = 0.57$, which is notably more complex when compared to logistic regression implementations ($s = 1.82$) and CS1 first week homeworks ($s = 2.67$) [26]. On the other hand, the Pyramid *image outputs* have Zipf distribution with exponent $s = 0.78$ (Figure 2b), suggesting that modern tools for understanding images may be more effective than an AST-based approach for understanding functionality of intermediate solutions.

3.2 Labeling milestones efficiently

The PyramidSnapshot dataset comes with *milestone labels*; that is, each image is categorized into one of our 16 visual categories of intermediate work. Figure 2a shows an example image for each of these milestones. Our milestone labels are mutually exclusive by design: an image must be either a scalene triangle or a pyramid-like image object, for example. We understand that there could be potential overlap in these milestones, but we chose exclusive labels because separable labels enable easier, clearer analysis. It is worth keeping in mind that there is a huge variation of images within each category; for example, Milestone 15 (Off-track) was marked for any image that was unable to be categorized by the other labels.

It would be an insurmountable task by a team of three researchers to label over 101,000 images in our PyramidSnapshot dataset. However, we observe from the Zipf distribution of the data (Figure 2b that only 27,220 (27%) of these are unique. Furthermore, the perfect pyramid (Milestone 13) in our dataset is the most frequent and occurs over 11,000 times across all student work, and the ten most popular images cover 20,219 images (20%) in our 101,636 image dataset. Using the Zipf distribution, one can label unique images to milestones in order of popularity in the dataset; a single researcher labeled 12,077 unique images with corresponding milestones in 20 hours over three days, covering 84,127 images (83%) of the actual dataset (Figure 2d). The *effort score* in Figure 2d characterizes the gain of labeling a unique image as the frequency of that image appearing in the overall dataset; the average effort score of unique images labeled was about 7 repeated images. On the other hand, each remaining unlabeled unique image appeared on average 1.2 times in the dataset, meaning that we would not have gained much with this labeling strategy had we continued into the tail of the distribution. Figure 2c shows that about 90% of students have at least 75% of their repositories labeled after using this strategy of labeling more popular images first.

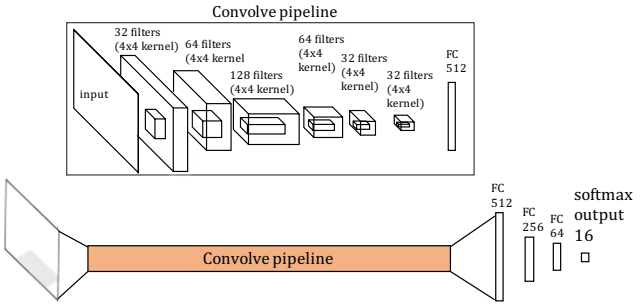


Figure 4: The neural network for milestone classification.

4 METHOD

We realize that assigning intermediate images to milestones presents a scalability issue for other assignments, even if we use image frequency to reduce labeling effort. We therefore explore how the accuracy of a classification model works as we vary N, the number of most popular images that we label, defined as our *training set* size. In this section, we define potential models for classifying intermediate snapshots by milestone, and we also outline an approach to answering a secondary question—whether classifying image output is sufficient for grading final submissions.

4.1 Classifying milestones

Given a training set of the N most popular images, we consider three models for classifying our images with milestones. Our first approach is based on the premise of unit testing: select one representative image for each milestone, and for each test image, predict a milestone only if the test image pixels exactly match that of the reference; otherwise it predicts nothing. We keep $N = 15$ unit testing images; we do not save a representative image for the Off-track milestone (Milestone 15). The second approach, K-nearest neighbors (KNN), is a common baseline used for computer vision [14]; for each test image, define the K nearest neighbors as the K images from the reference training set that have minimum sum-squared-difference (pixel-wise), and predict the most common milestone out of the nearest neighbors. Our third approach is a deep learning approach: train a convolutional neural network (Figure 4) to return the most likely milestone label based on processing a grayscaled, downsampled image of dimension $346 \times 477 \times 1$ with a 1.2-million parameter model. The model parameters are trained by optimizing a softmax cross-entropy loss over the N most popular images.

4.2 Extending classification to final solutions

In parallel with our goal to capture functionality of intermediate work, we also want to study whether an image-based, deep-learning approach is effective for autograding final submissions. From anecdotal experience, graphics assignments tend to be more difficult and time-consuming to grade than their command-line counterparts, mainly because the set of possible correct graphical outputs is often not well-defined. To grade our Pyramid assignment, a grader runs student final submission code with five predetermined sets of parameters which vary the brick size and Pyramid dimensions, then evaluates the student on a rubric set of 9 items, where for each item the grader marks a binary pass/fail by looking at both the student's code and five test results. We compare two approaches; the first is a unit-test-based model, where we have one counterexample image per rubric and we compare pixel-wise for exact match of output images. The second is a convolutional neural network approach with a similar structure to the milestone predictor in Figure 4; we train one pipeline for each of the five test output images and replace the classification layer with a sigmoid binary predictor per rubric; our model has 5.1 million parameters. We use a slightly different (non-public) dataset of 4383 Pyramid final code submissions with grades between 2009 and 2017.

5 RESULTS

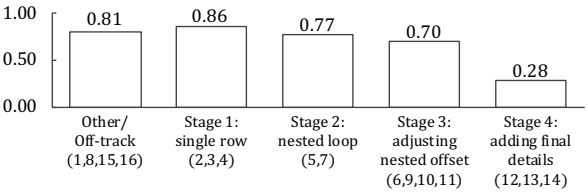
To evaluate our three approaches to classifying milestones from graphical image input, we trained each classifier with a varying size of N on our training set of the most popular images, and we evaluated its overall accuracy on two datasets: the *validation set*, composed of the top 11,000 popular images (corresponding to 70% of our image dataset), and the *tail set*, composed of the remaining 1,077 labeled images. We use the validation set to decide which classifiers work for common snapshots; the tail set accuracy is an indicator for whether our classifiers work on rarely seen snapshots.

We first discuss accuracy results on the validation set. The results for each classifier with the best choice of training set size N is reported as overall accuracy in classifying all 16 milestones in Figure 5a; we can immediately see that our neural network outperforms the other two unit-test and KNN ($K = 100$) models. We also report a higher-level statistic of overall accuracy over *knowledge stages*, which reduces our 16-dimension milestone space into five meaningful stages of knowledge, organized based on whether students are working on a single loop (Stage 1), a nested loop (Stage 2), adjusting brick offset within the nested loop (Stage 3), or completed with the assignment and going beyond what is expected (Stage 4). The remaining four stages are grouped as "Other/Off-track." We note that higher-level abstracted knowledge stages are more meaningful for a human grader as such stages will not differentiate between two milestones associated with the same level of student cognitive understanding. In Figure 5b we report our neural network accuracy for predicting a milestone that is in the same knowledge stage as the correct milestone label; the model reports at least 70% accuracy for Stages 1, 2, and 3, which are the most formative ones in terms of determining student progress towards the assignment goal, and therefore the low accuracy in identifying Stage 4 is acceptable. However, we realize that there is room for accuracy improvement across all knowledge stages for future work.

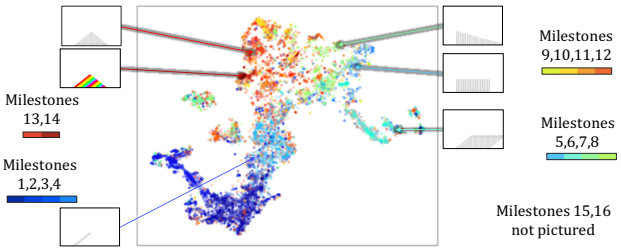
To understand how the model connects image to label, we use the t-SNE algorithm to compress the model's internal image representation into a 2-D, clustered visualization. Figure 5c shows that our image embeddings roughly cluster by their milestone; however, there are some perfect pyramid images (Milestone 13, red) hidden among the Hello world milestones (Milestone 1, blue). Upon checking these incorrectly classified images, we found that some of them depicted a very small pyramid, which could easily be misclassified as a single block, but others were simply incorrectly placed in the embedding space. Our hypothesis is that since unique images of single bricks dominate our dataset, our model has a tendency to predict Milestone 1 in the absence of any other strong indicators.

Overall accuracy	Unit test (N = 15)	KNN (K = 100, N = 11000)	Neural network (N = 11000)
By milestone (16 milestones)	.275	.037	.562
By knowledge stage (5 stages)	.275	.552	.649

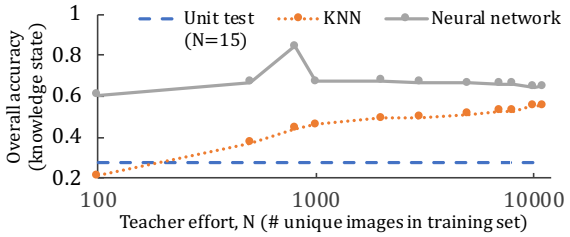
(a)



(b)



(c)



(d)

Figure 5: Milestone classification results. (a) Accuracy of each model (with training set size N) by milestone and by knowledge stage on the 11,000 most popular images. A closer analysis of the neural network model (N = 11000) shown by (b) Accuracy breakdown by knowledge stage, and (c) t-SNE plot of model embeddings, color-coded by milestone. (d) Overall accuracy (by knowledge stage) of KNN and neural network models with varying training set sizes N.

	Unit test image		Neural network	
	F1	Acc	F1	Acc
Train	0.15	0.95	0.65	0.97
Test	0.15	0.95	0.50	0.96

Figure 6: Average accuracy and F1 scores when predicting rubrics on final submissions.

We realize that it may seem intractable to extend our approach to other graphics-based assignments due to the sheer *effort* required to label the intermediate images for training data. We use the labeling scheme from Section 3.2 to create a training set consisting of the N most popular images; we then evaluate the performance of the KNN and neural network models with different training set sizes N, and we evaluate their performance on the top 11,000 most popular images (Figure 5d). Naturally, the KNN model performs best with a huge dataset, but what is surprising is that the neural network already outperforms the other two models even with only N = 100 items in the training set. There is a small peak at N = 800 for the neural network model; this artifact is because the model correctly predicted the milestones of some high-rank images, therefore weighting the accuracy upwards; this trend goes away when we compare accuracy across unique images only. In general, the takeaway that we can get just over 60% accuracy with just N = 100 labeled data points is very promising.

We also evaluated our images solely on the tail images in the distribution, the 1,077 least popular (labeled) images in the tail set. All three models performed poorly: Unit test reported 0% accuracy (due to its pixel-based exact match strategy), and both KNN and our neural network model performed with less than 1% accuracy for almost all choices of our training set. This is a huge area in which future models can improve; the more reliable we are in predicting the tail of our model, the more confident we can be in imputing milestone labels on unlabeled images in our dataset.

Final submissions evaluation. We report in Figure 6 our results of running our two final submission grading models. The average F1 scores and accuracies are evaluated on a validation set of 438 submissions; the neural network was trained on a set of 3945 submissions, and the unit-test model had 7 exact match reference images. Both models have very high average accuracies; however, this is because very few students get marked off for any of the 9 items in the rubric (the average frequency of incorrect rubric items was 10%). We therefore prioritize the F1 score metric, which is a harmonic mean of the precision and recall scores (measuring false positive and false negatives). We notice that the F1 scores for our neural network model are low for two reasons: first, several rubric items were code-based student misconceptions (e.g., incorrectly indexing either the inner or outer loop of a two-level nested loop, leading to indistinguishable final image outputs). Second, the rubrics marked for two different students with the same set of five unit-test images could vary widely. For example, for the 1485 student solutions that had the correct image output for all five brick configurations, there were at least 20 different rubric assignments, some with at most three rubric items marked off. Our image output-based approaches are not reliable enough to grade CS1 graphics assignments, but they are useful for labeling intermediate milestone work.

6 DISCUSSION

By splitting our milestone classification task into two phases—human labeling and deep-learning-facilitated labeling, we are able to achieve modest success on predicting the correct milestone label for any image. We use our milestone information to gauge how much time a student spends on average in each of our knowledge stages to decide which content to emphasize in classroom teaching.

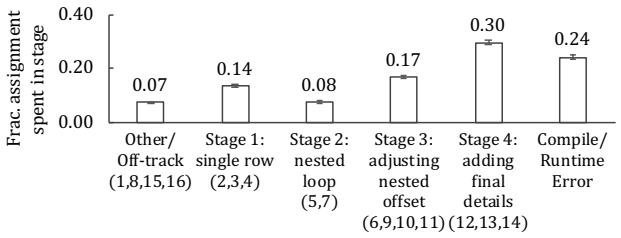


Figure 7: Average amount of assignment (by number of commits) that students spent in different knowledge stages.

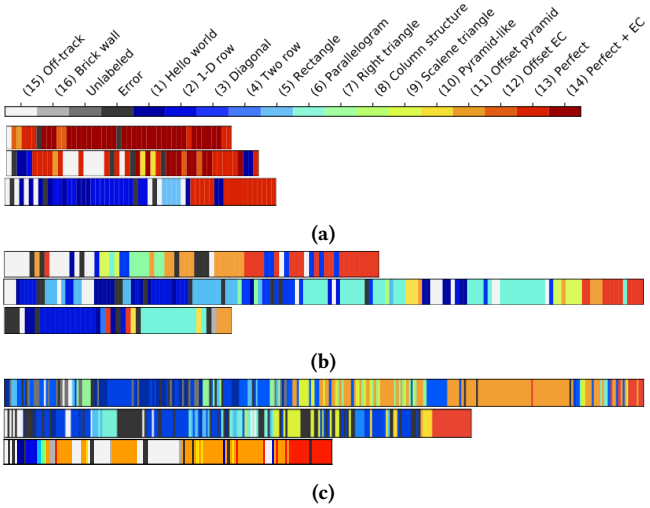


Figure 8: Student work patterns during the Pyramid assignment; a student's work is displayed as a horizontal strip over time, where each vertical slice is one snapshot. Three students respectively scoring in the (a) 99th percentile, and (b) 3rd percentile or lower on the midterm exam; (c) three students with long trajectories of work. Best viewed in color.

Figure 7 graphs the mean (±1 SE) fraction of commits spent in each of the knowledge stages, where we use our neural network predictor to *impute* the knowledge stages of the remaining 12% of the dataset, and the number of commits student spends each stage before moving on. We observe that on average, students spend 17% of their time in Stage 3, where they must manipulate the loop index to correctly offset blocks in different rows of the Pyramid. Understandably, students spend most of their time (30%) adding final details and finishing up (Stage 4). Knowing this time distribution can inform which concepts students struggle with; after this analysis, instructors devoted more time in class to discussing loop index manipulation skills needed for Stage 3.

We can also use milestone information to get a holistic view of students' functional progress over the Pyramid assignment; we show this via a case study of students based on their milestone traversal behavior. In Figure 8, we impute labels and visualize student work over time as a heatmap that indicates when students are in Stage 1 or 2 (blue colors), Stage 3 (yellow and green), or Stage 4 (red). After imputing the milestones on the remaining unlabeled snapshots, we found some very telling examples of how different students work on the assignment over time. When we compare high-performing students (Figure 8a) with low-performing (Figure 8b), we observe that these groups of students tend to concentrate their work in different stages. All three high-performing students have few snapshots in Stages 2 and 3, instead spending a significant portion of their time working on the perfect pyramid (Milestone 13). In contrast, the low-performing students spend a large fraction of time in Stage 3's early milestones that emphasize brick offset adjustments. One student also fails to reach the correct solution, ending instead in the offset pyramid milestone (Milestone 11).

From our data, we can also visually discern between certain students who were struggling to get anything working and those who were tinkering [5, 31]—where a student spends a long time at a particular knowledge stage not because they are stuck, but because they are exploring the solution space. Figure 8c shows three students that have used over 100 snapshots for the Pyramid assignment. The first two students spend a large portion of their time working in Stage 1. In contrast, the last student spends most of their time in a late Stage 3 milestone, the offset pyramid (Milestone 11), suggesting some sort of tinkering and adjustment. When we connect these work patterns with their performance on the midterm, the first two students score in the 19th and 21st percentile, respectively, while the third student scores in the 73rd percentile.

7 FUTURE WORK AND CONCLUSION

Our anecdotal analysis of this dataset suggests there is merit to observing snapshot histories with respect to milestone evolution. Having functional labels for snapshotted student data can be used in conjunction with existing indicators of progress, like error messages and code length [4, 5, 12], to predict student performance. As future work, we can cluster students and identify in aggregate which work patterns are correlated with stronger student performance. Future classrooms may evaluate student process in addition to final submission correctness; the color-coded heat map characterizing functionality over time provides a quick reference for instructors to detect which snapshots are pivotal for student learning. Towards this end, we have deployed the image label predictor as part of an in-classroom tool that allows teachers to interact with students and discuss the meta-process of how they program [34]. In this work, we have only scratched the surface of pedagogical insights that can be gleaned from more detailed analyses of student progress; by publishing our dataset, we hope that other researchers can improve upon our work to better understand how students code.

We realize that labeling intermediate snapshots is a human effort that is time-intensive for teachers and researchers. To see whether our approach was tractable for other assignments, we analyzed the tradeoffs between human labeling the most popular images and subsequent neural network-based labeling of remaining images. In our study, a simple neural network model trained on the 100 most popular images performs relatively well, suggesting that labeling for functionality is not as daunting as it may seem. Yet our biggest contribution is publishing the labeled PyramidSnapshot dataset used in this work as a new, annotated benchmark with two benefits to the community: to evaluate autonomous prediction for functionality of intermediate student program output, and to analyze student work patterns with greater detail than before.

REFERENCES

[1] Alireza Ahadi, Raymond Lister, Heikki Haapala, and Arto Vihavainen. 2015. Exploring Machine Learning Methods to Automatically Identify Students in Need of Assistance. In *Proceedings of the Eleventh Annual International Conference on International Computing Education Research (ICER '15)*. ACM, New York, NY, USA, 121–130. https://doi.org/10.1145/2787622.2787717

[2] Brett A. Becker. 2016. A New Metric to Quantify Repeated Compiler Errors for Novice Programmers. In *Proceedings of the 2016 ACM Conference on Innovation and Technology in Computer Science Education (ITiCSE '16)*. ACM, New York, NY, USA, 296–301. https://doi.org/10.1145/2899415.2899463

[3] Matthew Berland, Taylor Martin, Tom Benton, Carmen Petrick Smith, and Don Davis. 2013. Using Learning Analytics to Understand the Learning Pathways of Novice Programmers. *Journal of the Learning Sciences* 22, 4 (2013), 564–599. https://doi.org/10.1080/10508406.2013.836655

[4] Paulo Blikstein. 2011. Using Learning Analytics to Assess Students' Behavior in Open-ended Programming Tasks. In *Proceedings of the 1st International Conference on Learning Analytics and Knowledge (LAK '11)*. ACM, New York, NY, USA, 110–116. https://doi.org/10.1145/2090116.2090132

[5] Paulo Blikstein, Marcelo Worsley, Chris Piech, Mehran Sahami, Steven Cooper, and Daphne Koller. 2014. Programming pluralism: Using learning analytics to detect patterns in the learning of computer programming. *Journal of the Learning Sciences* 23, 4 (2014), 561–599.

[6] Adam S. Carter, Christopher D. Hundhausen, and Olusola Adesope. 2015. The Normalized Programming State Model: Predicting Student Performance in Computing Courses Based on Programming Behavior. In *Proceedings of the Eleventh Annual International Conference on International Computing Education Research (ICER '15)*. ACM, New York, NY, USA, 141–150. https://doi.org/10.1145/2787622.2787710

[7] Wanda P. Dann, Stephen Cooper, and Randy Pausch. 2008. *Learning To Program with Alice* (2 ed.). Prentice Hall Press, Upper Saddle River, NJ, USA.

[8] Stephen H. Edwards and Manuel A. Perez-Quinones. 2008. Web-CAT: Automatically Grading Programming Assignments. In *Proceedings of the 13th Annual Conference on Innovation and Technology in Computer Science Education (ITiCSE '08)*. ACM, New York, NY, USA, 328–328. https://doi.org/10.1145/1384271.1384371

[9] Mark Everingham, Luc Gool, Christopher K. Williams, John Winn, and Andrew Zisserman. 2010. The Pascal Visual Object Classes (VOC) Challenge. *Int. J. Comput. Vision* 88, 2 (June 2010), 303–338. https://doi.org/10.1007/s11263-009-0275-4

[10] David Hovemeyer, Arto Hellas, Andrew Petersen, and Jaime Spacco. 2016. Control-flow-only abstract syntax trees for analyzing students' programming progress. In *Proceedings of the 2016 ACM Conference on International Computing Education Research*. ACM, 63–72.

[11] Jonathan Huang, Chris Piech, Andy Nguyen, and Leonidas Guibas. 2013. Syntactic and functional variability of a million code submissions in a machine learning mooc. In *AIED 2013 Workshops Proceedings Volume*. 25.

[12] Matthew C. Jadud. 2006. Methods and Tools for Exploring Novice Compilation Behaviour. In *Proceedings of the Second International Workshop on Computing Education Research (ICER '06)*. ACM, New York, NY, USA, 73–84. https://doi.org/10.1145/1151588.1151600

[13] David S Janzen and Hossein Saiedian. 2006. Test-driven learning: intrinsic integration of testing into the CS/SE curriculum. In *ACM SIGCSE Bulletin*, Vol. 38. ACM, 254–258.

[14] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. 2012. ImageNet classification with deep convolutional neural networks. In *Advances in neural information processing systems*. 1097–1105.

[15] Lucas Layman, Laurie Williams, and Kelli Slaten. 2007. Note to Self: Make Assignments Meaningful. In *Proceedings of the 38th SIGCSE Technical Symposium on Computer Science Education (SIGCSE '07)*. ACM, New York, NY, USA, 459–463. https://doi.org/10.1145/1227310.1227466

[16] Yann LeCun and Corinna Cortes. 2010. MNIST handwritten digit database. (2010). http://yann.lecun.com/exdb/mnist/

[17] Colleen M. Lewis. 2012. The Importance of Students' Attention to Program State: A Case Study of Debugging Behavior. In *Proceedings of the Ninth Annual International Conference on International Computing Education Research (ICER '12)*. ACM, New York, NY, USA, 127–134. https://doi.org/10.1145/2361276.2361301

[18] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. 2013. Playing Atari with Deep Reinforcement Learning. *arXiv preprint arXiv:1312.5602* (2013).

[19] Bassam Mokbel, Sebastian Gross, Benjamin Paassen, Niels Pinkwart, and Barbara Hammer. 2013. Domain-independent proximity measures in intelligent tutoring systems. In *Educational Data Mining 2013*.

[20] Briana B. Morrison, Lauren E. Margulieux, and Mark Guzdial. 2015. Subgoals, Context, and Worked Examples in Learning Computing Problem Solving. In *Proceedings of the Eleventh Annual International Conference on International Computing Education Research (ICER '15)*. ACM, New York, NY, USA, 21–29. https://doi.org/10.1145/2787622.2787733

[21] Lili Mou, Ge Li, Lu Zhang, Tao Wang, and Zhi Jin. 2016. Convolutional Neural Networks over Tree Structures for Programming Language Processing. In *AAAI*, Vol. 2. 4.

[22] Nick Parlante, Steven A Wolfman, Lester I McCann, Eric Roberts, Chris Nevison, John Motil, Jerry Cain, and Stuart Reges. 2006. Nifty assignments. In *ACM SIGCSE Bulletin*, Vol. 38. ACM, 562–563.

[23] Steven T. Piantadosi. 2014. Zipf's word frequency law in natural language: A critical review and future directions. *Psychonomic Bulletin & Review* 21, 5 (2014), 1112–1130. https://doi.org/10.3758/s13423-014-0585-6

[24] Chris Piech, Jonathan Huang, Andy Nguyen, Mike Phulsuksombati, Mehran Sahami, and Leonidas Guibas. 2015. Learning Program Embeddings to Propagate Feedback on Student Code. In *Proceedings of the 32nd International Conference on Machine Learning (Proceedings of Machine Learning Research)*, Francis Bach and David Blei (Eds.), Vol. 37. PMLR, Lille, France, 1093–1102.

[25] Chris Piech, Mehran Sahami, Daphne Koller, Steve Cooper, and Paulo Blikstein. 2012. Modeling how students learn to program. In *Proceedings of the 43rd ACM technical symposium on Computer Science Education*. ACM, 153–160.

[26] Chris J. Piech. 2016. *Uncovering Patterns in Student Work: Machine Learning to Understand Human Learning*. Ph.D. Dissertation. Stanford University.

[27] Mitchel Resnick, John Maloney, Andrés Monroy-Hernández, Natalie Rusk, Evelyn Eastmond, Karen Brennan, Amon Millner, Eric Rosenbaum, Jay Silver, Brian Silverman, and Yasmin Kafai. 2009. Scratch: Programming for All. *Commun. ACM* 52, 11 (Nov. 2009), 60–67. https://doi.org/10.1145/1592761.1592779

[28] Eric Roberts. 2000. Strategies for Encouraging Individual Achievement in Introductory Computer Science Courses. In *Proceedings of the Thirty-first SIGCSE Technical Symposium on Computer Science Education (SIGCSE '00)*. ACM, New York, NY, USA, 295–299. https://doi.org/10.1145/330908.331873

[29] Stephanie Rogers, Dan Garcia, John F. Canny, Steven Tang, and Daniel Kang. 2014. *ACES: Automatic Evaluation of Coding Style*. Master's thesis. EECS Department, University of California, Berkeley. http://www2.eecs.berkeley.edu/Pubs/TechRpts/2014/EECS-2014-77.html

[30] Elizabeth Sweedyk, Marianne deLaet, Michael C. Slattery, and James Kuffner. 2005. Computer Games and CS Education: Why and How. In *Proceedings of the 36th SIGCSE Technical Symposium on Computer Science Education (SIGCSE '05)*. ACM, New York, NY, USA, 256–257. https://doi.org/10.1145/1047344.1047433

[31] Sherry Turkle and Seymour Papert. 1990. Epistemological Pluralism: Styles and Voices within the Computer Culture. *Signs: Journal of Women in Culture and Society* 16, 1 (1990), 128–157. https://doi.org/10.1086/494648

[32] Ke Wang, Rishabh Singh, and Zhendong Su. 2018. Dynamic Neural Program Embedding for Program Repair. *International Conference on Learning Representations* (2018).

[33] Lisa Wang, Angela Sy, Larry Liu, and Chris Piech. 2017. Deep Knowledge Tracing On Programming Exercises. In *Proceedings of the Fourth (2017) ACM Conference on Learning @ Scale (L@S '17)*. ACM, New York, NY, USA, 201–204. https://doi.org/10.1145/3051457.3053985

[34] Lisa Yan, Annie Hu, and Chris Piech. 2019. Pensieve: Feedback on Coding Process for Novices. In *Proceedings of the 50th ACM Technical Symposium on Computer Science Education (SIGCSE '19)*. ACM, New York, NY, USA. https://doi.org/10.1145/3287324.3287483

[35] Lisa Yan, Nick McKeown, Mehran Sahami, and Chris Piech. 2018. TMOSS: Using Intermediate Assignment Work to Understand Excessive Collaboration in Large Classes. In *Proceedings of the 49th ACM Technical Symposium on Computer Science Education (SIGCSE '18)*. ACM, New York, NY, USA, 110–115. https://doi.org/10.1145/3159450.3159490

[36] Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc V Le. 2017. Learning transferable architectures for scalable image recognition. *arXiv preprint arXiv:1707.07012* (2017).