Scaling Learner Feedback

Vineet Pandey
Computer Science and Engineering
UC San Diego
La Jolla, CA 92093-0404
vipandey@eng.ucsd.edu

ABSTRACT

Learners require feedback on their work to understand mistakes and find ways to improve. Providing feedback on creative technical or narrative work such as programs or essays is difficult since they have multiple success criteria and require assessing deep features. Massive Open Online Courses (MOOCs) amplify this challenge since instructor resources do not scale with the number of learners. However, MOOCs also provide a large set of learner submissions to model errors, a large set of peers to provide feedback to each other using instructor artifacts and a platform to run large-scale experiments to quickly evaluate the efficacy of different feedback techniques. In this paper, I survey research towards assessing creative artifacts at the scale of MOOCs, using automated tools, peers or both. First, I describe the challenges presented by MOOCs towards providing useful feedback to learners. Second, I present the state of the art techniques from machine learning, programming languages and crowdsourcing domains that provide feedback to learners at scale. Third, I synthesize specific principles underpinning these tools that are independent of their actual implementation. Finally, I lay the path forward for providing useful feedback to learners at scale by understanding learner needs and instructor constraints, and combining automated tools with instructor/peer feedback.

Author Keywords

Feedback; assessment; MOOCs; online education

ACM Classification Keywords

K.3.1 [Computer Uses in Education]: Distance learning, Collaborative Learning.

USEFUL FEEDBACK SERVES MANY LEARNING GOALS BUT IT IS DIFFICULT TO GENERATE

Learners require feedback on their work to understand mistakes and find ways to improve. Receiving feedback allows learners to understand the quality of their ideas and to correct their misconceptions [50]. Practice and feedback are essential to fix incorrect understanding while developing new skills [49]. Feedback provides a better idea of the expected standards and allows learners to compare their work to it [22].

Feedback can be classified in a number of ways including the learner objectives it hopes to meet (such as, reflection vs. quick fix), the different forms (such as, hints vs. counter-examples) and its performance on various features (such as, accuracy, coverage or latency). Appendix describes this feedback taxonomy in detail. This paper synthesizes principles based on the success and failure of current systems that provide feedback at scale, and identifies their limitations to propose future work.

Multiple Choice Questions: Canonical way to scale evaluation

Multiple Choice Questions scale well

Multiple Choice Questions (MCQs) have been used to scale assessment in examinations across the globe including standardized tests such as Scholastic Aptitude Test in USA (SAT, https://collegereadiness.collegeboard.org/sat/inside-the-test), or entrance admissions for undergraduate universities in China or India (such as Joint Entrance Examination for IITs, http://jeemain.nic.in/).

When assessing learner understanding, MCQs have multiple advantages. MCQs can be graded quickly and consistently. Due to the low latency of testing, quick tests can help with retaining knowledge by working on the retrieval path of the brain rather than just the encoding path [29]. Hence, quick multiple-choice quizzes in classrooms during instruction or while viewing online lectures helps learning. Moreover MCQs do not suffer from grading inconsistency since machines or people alike can evaluate them quickly.

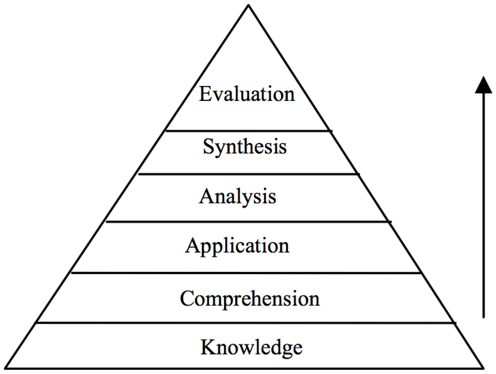


Figure 1: Bloom's levels of cognition showing increasing difficulty of learning tasks [9]

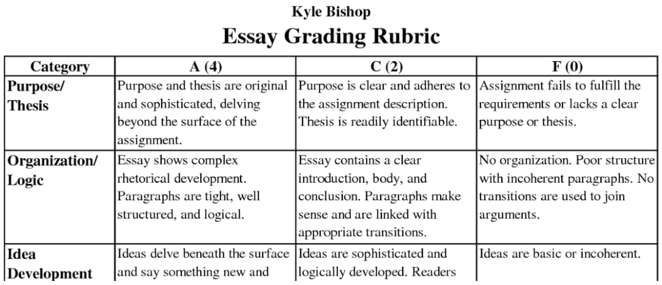
Multiple Choice Questions are difficult to design to test higher cognition like evaluation or synthesis

MCQs are difficult to design to test higher cognition tasks, such as evaluation/ synthesis, as shown in Bloom's taxonomy of human cognition in Figure 1. Learners tend to pick the right answer in MCQs using a number of techniques such as elimination (removing incorrect options) or recognition (identifying the right option rather than creating it). Moreover, designing good MCQs to know more than fact recall requires instructor effort. It requires focus on deeper rather than surface features, avoiding issues in wording or grammar to avoid throwing learners down wrong path, and generating careful multi-step questions to identify the depth of learner's understanding [7].

Formative feedback is difficult to produce with MCQs

MCQs typically provide feedback by telling if the answer is right or showing the right answer (perhaps with some description). Providing feedback by showing the right answer, while useful, has its limitations. In some domains, this feedback might be sufficient (e.g.: while testing knowledge retention, such as historical facts). In other tasks like solving a math problem to find the final answer, it does not provide sufficient insights for the learner to know where they went wrong (for example, which step of the integral calculation has an error).

Furthermore, in a variety of creative tasks that have more than a small number of correct or successful answers, it fails to account for learner's work. Showing the correct solution with the correct option tells the learner about the right way to solve the problem if they were following the same procedure, but in other cases, as is typical in creative exercises, it is difficult to receive useful feedback via even carefully designed MCQs.



Kyle Bishop

Essay Grading Rubric

Category	A (4)	C (2)	F (0)
Purpose/ Thesis	Purpose and thesis are original and sophisticated, delving beyond the surface of the assignment.	Purpose is clear and adheres to the assignment description. Thesis is readily identifiable.	Assignment fails to fulfill the requirements or lacks a clear purpose or thesis.
Organization/ Logic	Essay shows complex rhetorical development. Paragraphs are tight, well structured, and logical.	Essay contains a clear introduction, body, and conclusion. Paragraphs make sense and are linked with appropriate transitions.	No organization. Poor structure with incoherent paragraphs. No transitions are used to join arguments.
Idea Development	Ideas delve beneath the surface and say something new and	Ideas are sophisticated and logically developed. Readers	Ideas are basic or incoherent.

Figure 2: Sample Essay grading rubric showing 3 out of 10 features of an essay [8]. None of these features can be directly measured using automated tools, since they require language-processing skills currently not demonstrated by algorithms.

Open-ended assignments are valuable for learner learning

Design, programming and writing essays are examples of domains that require creative thinking and different learners produce unique submissions. Examples of design tasks include designing a webpage. Programming can include solving novice problems towards learning a particular language's features (e.g. https://www.coursera.org/course/programming1) or learning to program algorithms (e.g. https://www.coursera.org/course/algs4partI). Writing essays is a popular assessment technique to understand learner's skills beyond just framing grammatically correct sentences to combining facts, opinions and evidence among other features. Moreover, essays can have multiple styles, such as explanatory, exploratory or persuasive [55]. Design, programming and writing essays provide a large expanse of open-ended creative tasks that span a broad range of educational objectives, across design, computer science and humanities.

Richer assignments can test generative submissions

Questions with clear correct and incorrect answers provide a limited set of choices and identifying the correct solution does not imply that the learner can generate it [48]. Open-ended assignments require learners to generate solutions rather than only recognizing them, and can be a better check for performance on realistic tasks. Embracing multiple solutions is another useful feature of open-ended assignments. Assessment of open-ended work is also difficult due to its inherent variability and the challenge in creating a complete set of evaluation criteria [5].

Difficult to provide feedback on richer assignments

Assessing the quality of creative work such as creating technical artifacts (programming tasks) and narrative artifacts (essays) involves tackling multiple challenges.

Multiple success criteria: Technical artifacts like programs have multiple ways to be adjudged successful. For instance, despite producing the correct output, a program should not be slow or consume large resources. This is different from rules-based work like solving a math problem to get the answer, where there is straightforward application of rules to achieve one clear success criteria. Similarly in essays, it is difficult to mention one well-defined criterion that makes for a successful essay. Essays are judged based on their structure, the presentation of evidence towards the objective and the choice of words. Figure 2 provides some criteria for essay evaluation.

Multiple potentially successful solutions: Creative work is rarely bound by one standard solution. For any programming puzzle of sufficient complexity, different algorithms can be used with their own trade-offs. Similarly, code can be designed in many idiosyncratic ways. So, a code that performs poorly but is written in a readable manner might score over code that performs well but cannot be understood by people; however, none of them are unsuccessful. Moreover, multiple solutions can meet the same success criteria. For example, different essays, depending on writing style, can be persuasive to the same degree in using the evidence towards their thesis.

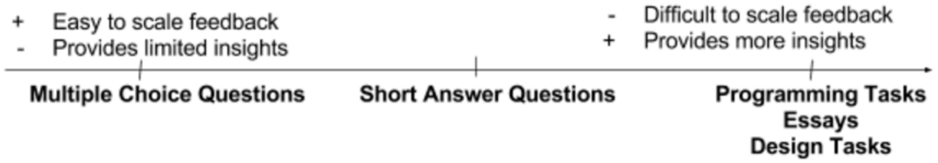


Figure 3: While richer assignments like programming tasks provide more insights into student learning, they are difficult to provide feedback on at scale

Difficulty in evaluating deeper features rather than surface features: Essays have a large number of features to be considered when compared to a standard numerical problem, such as calculating the area of a circle given the radius. While some of them are quantifiable, they are not a great measure of the quality of essay. Such surface features include number of words, average size of word, size of a paragraph, the actual words used. Some deeper features that describe an essay include the structure (how the different parts come together), flow (does one section organically lead to the next one), content (the actual ideas demonstrated helpfully in simple terms), presentation of ideas, and use of evidence in meeting the etc.

Despite these challenges, richer assignments like programming tasks and essays are used to test higher levels of cognition (as shown in Figure 1, originally from [9]). How can current techniques to assess such creative work in classrooms scale to orders of magnitude more learners?

SCALE IN MOOCS PROVIDES CHALLENGES AND POTENTIAL SOLUTIONS

In a classroom, instructor staff can assess essays or programs and provide feedback to learners. However, MOOCs have thousands of learners, which makes it impossible for the instructor staff to provide meaningful feedback by looking at every submission. Effort needed to scale learner feedback scales linearly with the number of learners while instructor effort does not. Techniques to provide feedback at scale amplify instructor knowledge by applying to learner submissions. However, there are still significant challenges around evaluating creative work like essays, programs or design. **How can learners receive feedback on creative exercises at the scale of MOOCs?**

Fortunately, scale provides a number of advantages as well. For HarvardX MOOCs, the mean enrollment was 43K learners, and the number varied from 550 learners in HarvardX Copyright class to 190K in HarvardX IntroCS class [39]. Such a large scale provides a large set of learner submissions, peers and a platform to test various ideas.

Large corpus of learner submissions: MOOCs provide a large number of learner submissions that can be used in many useful ways:

1. Large set of submissions can be used to identify common errors made by learners, which can be formalized as explicit error models for automated tools [52]
2. Learner submissions show how learners fix their errors or improve their work. This can be used to construct learner pathways, that can be used to help future learners get unstuck by identifying similar problem-solving strategies: [44,48]
3. Similar set of submissions can be used to provide differential feedback, by referring to another similar but better submission using clustering techniques [38]

Peers: Scale also provides a large set of peers who can provide feedback to each other after suitable training. Despite using automated tools to provide feedback in some domains like programming, there are still other domains and different aspects of programming for which useful automated tools do not exist. For example, subjective open-ended feedback on essays using algorithms is currently not possible based on Natural Language Processing research. While Machine Learning techniques like Latent Semantic Analysis [35] can be used to assign grades to different essays, it is not possible to find corrective or explorative feedback. However, instructors are trained to identify the issues in a specific submission, provide comments to rectify the issues and urge learners to expand the solution space. Can this training be passed to other learners in the class to provide useful feedback to other friends?

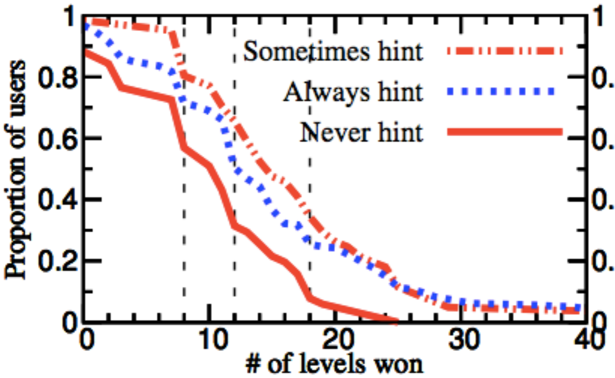


Figure 4: An example of how running user studies at scale can help us identify techniques that work. In a study where learners playing a game that tested their programming skills [40], never providing hints led to reliably poorer performance while surprisingly, sometimes providing hints performed better than always providing hints for most users. This provides further research questions into uncovering the reasons.

Platform to run quick, large-scale experiments: New ways of providing feedback should be tested for the actual effects they produce. For instance, a study [23] found that "Ordering of questions makes (1) no difference to the time taken to complete the examination, or (2) to the results, (3) however it may have an effect on learner attitude." The research used 127 learners across a semester and tested them with 30 questions. For online courses, the presence of large number of learners can allow quick testing of different feedback techniques to see if they genuinely help learners. Figure 4 shows one such attempt. Moreover, carefully designed experiments can be used to identify real issues in feedback usage, creating further research questions towards improving learning. Lomas et al. have proposed using online platforms as a way to iteratively build design theory around the efficacy of solutions [36]. For instance, can a new hint system adversely affect learner performance [41]? Moreover, the scale and diversity of MOOCs provide us an important opportunity to identify data-driven design principles and to contribute back to learning science research, thus providing a significant opportunity to build and test theory via experiments at scale.

How can learners receive feedback on creative exercises at the scale of MOOCs? The following sections of the paper discuss using properties of scale to build useful feedback tools and identify their limitations and opportunities for future work.

AUTOMATED FEEDBACK TECHNIQUES MATCH INSTRUCTOR INPUTS TO LEARNER SUBMISSIONS

Useful feedback uses two inputs intelligently. Firstly, it uses **learner submissions** to identify and fix specific issues rather than provide blanket suggestions. Given an incorrect program or an unsuccessful essay, it is useful for the learner to receive feedback on their submission to understand the errors or the low-level issues such as grammar or sentence construction. While it is useful for learners to know what they did wrong, it is important to steer them towards the right direction, since local fixes might sometimes not yield a global fix. Therefore, useful feedback uses **instructor inputs**. Instructors' domain knowledge and their understanding of learning objectives allows them to create solution templates and error models of learner work which can be combined with learner submissions to provide useful feedback.

Design space for feedback at scale

Feedback at scale has three important dimensions:

1. Inputs to the feedback system: Does it use instructor inputs and learner submissions? Moreover, is instructor input used before generating feedback or is it an interactive feedback process?
2. Output of the system: What is the form of feedback provided? For instance, it can be grades, hints, fixes or high-level feedback.
3. Techniques used: Does the system use automated tools based on Machine Learning or Programming Languages work which can be data-driven or inductive, or does it rely on crowd techniques, that use the people part of the system or crowds to provide feedback?

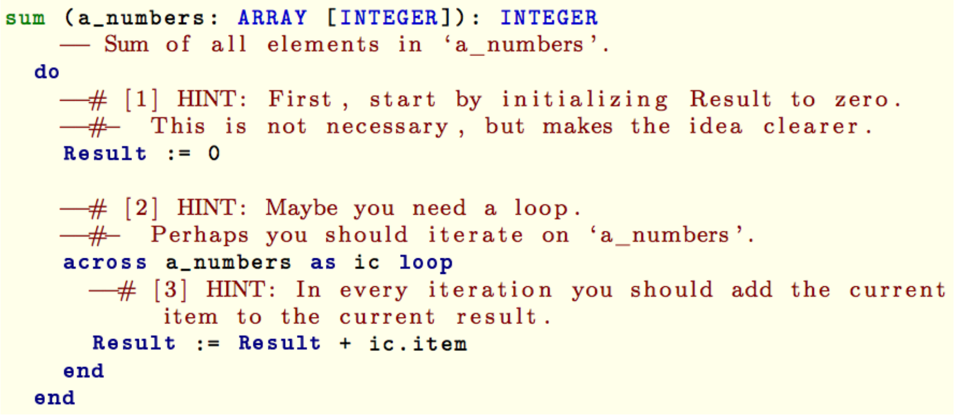
Nature of Feedback	**Technical tasks** **(Programming)**	**Writing/Design tasks** **(Short answers, essays)**
Grade	Based on test-cases	Latent Semantic Analysis [26], Peers [32]
Hints	AutoTeach [3], Test-driven synthesis [43]	-
Fixes	Synthesis based: Autograder [54], Automata Tutor [52, 2]	Crowds [6]
High-level feedback	Crowds [63], Cluster-then-instructor [11]	Peers [34]; Crowds [19]

Table 1: Systems providing different feedback on creative domains like programming or writing essays. Providing grades can be automated but providing high-level feedback requires humans in the loop

This taxonomy links the input to the output and explains the techniques that make this link. This design space is explained in Table 1.

Providing feedback on programming tasks is non-trivial

Consider a programming assignment where learners are required to sort an array. It is easy to assess whether their sorting algorithm works correctly: i.e. it meets the purpose or *function* based on the results on test set. However, providing useful feedback about why an incorrect program struggles on some inputs requires better understanding of the program. Providing failing test cases is not too helpful because new learners need to map it to lines in their code. Learners seek help on discussion boards, but the responses are slow. Also, novice feedback might not be specific or complete - leading to further queries and wait. Automated bug fixing does not help either because it is local (checks against test suite or partial spec.



```
sum (a_numbers: ARRAY [INTEGER]): INTEGER
    -- Sum of all elements in 'a_numbers'.
  do
    --# [1] HINT: First, start by initializing Result to zero.
    --#- This is not necessary, but makes the idea clearer.
    Result := 0

    --# [2] HINT: Maybe you need a loop.
    --#- Perhaps you should iterate on 'a_numbers'.
    across a_numbers as ic loop
      --# [3] HINT: In every iteration you should add the current
            item to the current result.
      Result := Result + ic.item
    end
  end
```

Figure 5: Hello world program annotated with textual hints that show when a learner makes a mistake in a hint's vicinity [3]. These hints, manually typed in by the instructor, do not verify whether the hint is related to learner error.

Instructor assumptions about learning breakdowns can be wrong if actual learner data is not consulted

AutoTeach [3] is a simple hint mechanism where instructor encodes textual hints in the program. As a learner make errors, the hints in the vicinity of the error are shown, as shown in Figure 5. If hints are not helpful then portions of the code are made available. This approach does not use the specific error made by a learner since it provides a generic hint for errors around a location that does not depend on their content. Moreover, it uses instructor inputs that are either textual hints explicitly coded by the instructor in the program rather than using instructor knowledge towards creating a general error model that might be applicable to different parts of the current (or any other) program.

The results for this system are not promising since 1/188 (<1%) submission without using hints created incorrect solutions while 53/143 (<37%) learners using hints created incorrect solutions. Moreover, since an instructor manually adds hints to the solution, it cannot scale for more than a few problems, and as shown above, instructors might not know the right location or hints needed without consulting actual data showing learner errors.

Actual learner pathways can be used to generate hints

Large scale of submissions can be used to find pathways from the initial state of a learner's submission to the final answer [48, 44]. Using 60K+ submissions from Hour of Code submissions, learner paths were drawn to identify how the code progress from an empty document to the correct result. For learners stuck at a particular part of the program, hints can be provided about the next state by looking at the path that leads to the solution. Moreover, this path construction can be done based on a number of factors such as the shortest time taken to reach the final correct solution or the shortest number of states in the process. One such set of paths is shown in Figure 6.

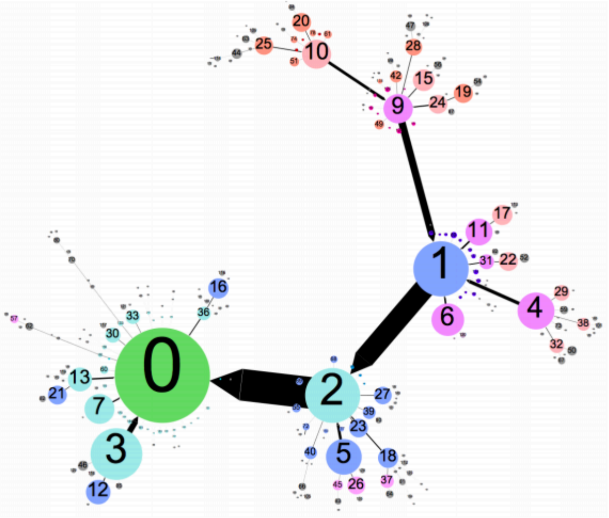


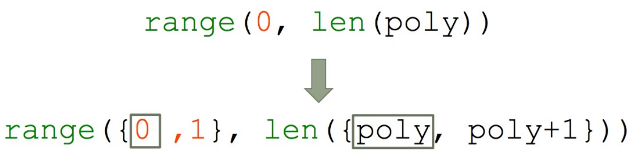
Figure 6: Visualization of problem-solving policies in a Code of Hour challenge [42]. Node 0 is the correct answer (and the end goal) while other nodes are unique partial solutions. Edges show potential paths from partial solutions to the final answer.

While this work deals with code snippets from a trivial coding challenge for novices, there exist challenges with real-world programming assignments. Larger programs will have larger set of states and many more paths mapping to problem-solving policies. Moreover, due to diversity in solutions, clearly popular paths might be absent. Moreover, this approach does not utilize instructor inputs and the popular path to the final answer might not be best suited to learning objectives. Instructor involvement in an interactive way that stops part of this path might be useful.

Error models can be built using scale of submissions

In the Autograder work [54], the problem of providing feedback on the wrong parts of a code is reduced to the problem of generating the correct program from an incorrect program using minimum number of changes. Synthesizing the correct program for a given task is a harder problem than fixing a broken program and this reduction works due to the error model provided by the instructor that provides a large set of candidate programs that need to be tested for equivalence with the correct program. Effectively, the problem of synthesizing the correct program is converted to choosing the correct program from the multiple syntheses. Autograder uses both instructor inputs and learner submission to generate program fixes.

The rule-based translation strategy, shown in Figure 7, for synthesis provides useful feedback opportunities because it explicitly identifies the rules that lead to the functionally equivalent program to the correct program. Therefore, it can provide explicit line/sub-expression specific feedback on the program that is otherwise difficult. Autograder feedback includes the location of error (line number), the problematic expression in the line and the sub-expression to be modified and its new value. However, the key challenge with this work is the limited problems captured in learner submissions. For Autograder, 36% of the programs in the dataset had big conceptual errors that could not be fixed by the tool, implying the error model can potentially help with local fixes but globally wrong solutions cannot make use of this approach.



```
range(0, len(poly))
range({0 ,1}, len({poly, poly+1}))
```

Figure 5: Sample rewriting that demonstrates an off-by-one error stored in the error model [51]. Given the first line, the system converts it to all possible off-by-one variants (2 * 2 = 4 options) and checks all of them.

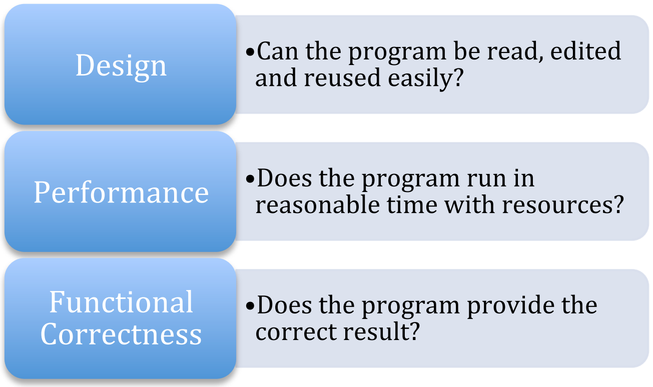
A framework of synthesis based feedback techniques

Autograder is an instance of a general class of synthesis/ classification based feedback tools, which also includes CPS grader [28] and Automata Tutor [2]. All these tools follow a common three-step process that differs in the underlying implementation, described in detail in Appendix.

1. **Preprocess step**: Use instructor expertise to build a library of common mistakes, a reference implementation for solution and to precompile meta comments in a markup language
2. **Encode step**: Convert the domain/problem to a language
3. **Generate step**: Use program synthesis/ ML techniques to generate feedback

Writing programs involves more than functional correctness

A program must provide the right result. However, factors such as performance or design of code play a big role in its longevity by deciding whether it needs to be improved and whether it will be reused.



Design
- Can the program be read, edited and reused easily?

Performance
- Does the program run in reasonable time with resources?

Functional Correctness
- Does the program provide the correct result?

Figure 6: Providing feedback on programs goes beyond just functional correctness. How can automated tools and people provide useful feedback on performance and design of programs?

A study found that 60% of functionally correct programs had performance problems [24]. This is unsurprising since the primary goal of novice programmers is to ensure their code runs produces the right result. However, a narrow focus on somehow getting the right result can lead to poor coding practices, in terms of performance of the code as well as its design. A good software engineer ensures that his/her code runs on machines in reasonable time with reasonable amount of resources. Since a majority of online learners doing programming courses are motivated to become software engineers (https://www.insidehighered.com/news/2012/06/05/early-demographic-data-hints-what-type-student-takes-mooc), it is even more important to train them well in basic practices of software engineering, that go beyond simple functional correctness.

Performance correctness: Identify abstract strategies and specific solutions, and provide feedback accordingly

Many domains have a solution strategy that involves two phases: coming up with an abstract strategy to solve a problem and then fixing the specific details of how to achieve that solution strategy. This has shown to be true in writing math proofs, programming or even writing research papers. In terms borrowed from Design of Everyday Things [40], they can be called plan and implementation respectively. For programming tasks, a plan corresponds to finding a strategy that uses one or more algorithms to produce the result, while the implementation would include specific instantiations of the algorithm.

Performance of a program is determined by both the plan as well as the implementation. Current work [24] focuses on identifying the plans in a program and providing corrective feedback. For instance, suggesting using binary search in place of linear search would be corrective performance feedback. This involves trace-analysis of the program by observing the intermediate values observed after annotating variables in the program. The algorithmic strategy employed in a solution can be observed from the values generated during execution of the program. While these algorithmic strategies can be a large number in a general case, for simple programming questions among novices, they are a small number and can be tackled by an automated system.

Trace-based analysis, unlike source-analysis of a program, does not require conversion to an intermediate language, since it does not depend upon syntactic details. However, novel but uncommon solutions might not be captured correctly by traces, hence, there should be a way to add them to the actual abstract solution library. One line of enquiry could be to combine source-code analysis and trace-based analysis to infer both problems at plan and implementation levels.

In previous systems, instructor input was used to construct template(s) [3, 54, 24], learner pathways and error models [3, 54,1]. **Automated tools for providing feedback on programming tasks scale instructor knowledge.**

FEEDBACK ON WRITING TASKS REQUIRES HUMANS IN THE LOOP

Essays are a popular tool to assess to assess learners on their ability to use facts, arguments and language features towards exposition of a topic or towards persuading a compelling point of view on a subjective question. Essays are a good test of higher levels of cognition that include analysis and synthesis. Compared to Multiple Choice Questions, they are easier to construct towards testing these aspects of learning. A successful essay demands successful use of purpose, content, organization, evidence, audience, content, style, format and grammar, among others. However, this makes assessment of essays a non-trivial activity.

Identifying features like use of evidence or organization of an essay requires skills on part of the evaluator to be able to construct these features by identifying their signs. This is non-trivial for people and algorithms are far from making such deep judgments of essays. Therefore, to enable people to assess essays correctly, the assessment criteria are made explicit using rubrics such as one shown in Figure 2. The analogue of such rubrics does not exist for algorithms. Essays are currently graded successfully using Latent Semantic Analysis [35] despite its failure in breaking an essay into its constituent deeper features. Appendix describes this technique in detail.

Latent Semantic Analysis technique can grade essays but not provide specific feedback

Latent Semantic Analysis (LSA) clusters essays with similar knowledge content together using a number of features that act as measures of information content of an essay. Figure 9 shows how LSA is used in EdX to grade essays. Previous studies have found LSA to find similar correlations to human raters [cite intro LSA] as humans themselves. Moreover, like other automated techniques, LSA does not suffer from the problems of grading fatigue and inconsistency.

LSA techniques cannot provide formative feedback

Despite its relative success and advantages, its algorithms can be fooled into assigning high grades to seemingly gibberish essays [66] if the features it relies on are known apriori. In short, LSA fails at developing an understanding of the domain of essay as well as its proficiency in strongly providing key ideas and arguments to make for a great essay. Moreover, there are a number of reasons specific to the technique that makes it incapable of providing open-air feedback.

Intra-cluster variance: After putting items in the same cluster, it is difficult to ascertain the quality of individual items. Moreover, at a large scale of submissions, the error rates in clustering might be high.

Lack of evaluation of presentation: LSA does not use order of words as a parameter while developing clusters. This implies that essays containing similar information can vary a lot in their presentation and readability, which are important parts of essay

Assumptions about underlying distributions: LSA assumed Gaussian distribution for words in a document and assumed Frobenius norm for calculating distance between items in the final n-dimensional space. There is no evidence that these are the perfect parameters for grading essays.

Other statistical techniques rely on textual features rather than semantic understanding by using a combination of features like using bigrams/trigrams, syntactic parsing and dimensionality reduction.

Can automated tools provide feedback on essays?

Learners would gain most from receiving feedback on their

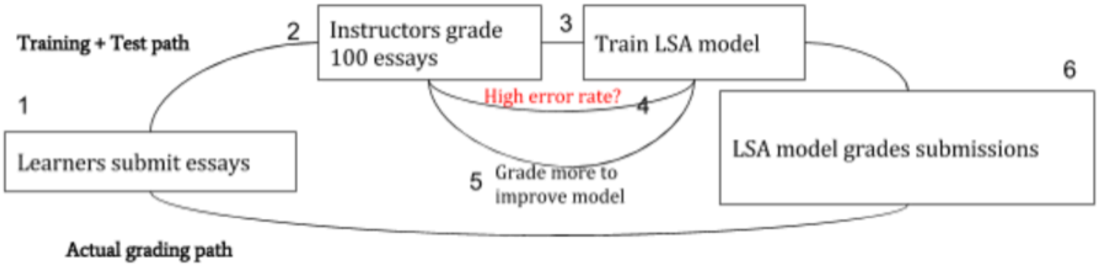


Figure 7: Instructor graded essays allow Latent Semantic Analysis (LSA) model to identify grades for other essays by putting them in clusters with the same information content as an already graded essay.

essays around structure, arguments and evidence. Latent Semantic Analysis or other statistical techniques fail to provide feedback around those issues. “Understanding” the meaning of a text is a fundamental Natural Language Processing problem and without drastic algorithmic improvements, automated tools would need to work with humans in the loop to provide useful feedback. For example, crowds can be used with grading tools such that lower-graded essays can be passed to trained peers/crowds who can provide more specific feedback. Another approach could be to grade the separate deeper features of an essay to help isolate the dimensions that some essays perform poorly on. However, it is difficult to separate content, presentation and user’s choice of linguistic features, rendering this approach wildly optimistic at best.

Clustering similar submissions amplifies the reach of instructor feedback

While LSA does not provide feedback, its idea of clustering similar submissions can be used with instructor feedback to provide high-level feedback at a large scale with chances of errors. Divide and correct approach [11] clusters *short answer submissions* using powergrading algorithm [4]. Instructors grade a representative member of a cluster and this grade is assigned to all members of the corresponding cluster. Moreover, instructors can provide feedback to clusters that might be relevant for many submissions in the cluster. Figure 10 shows the Divide and Cluster interface that allows instructors to achieve this goal.

A within-subjects study with 25 participants with prior teaching experience graded 200 distinct answers from two questions each found that clustering interface reduced the time of grading by an order of magnitude as compared to a flat interface where instructors used went through submissions sequentially. The reduction in grading time did not come at the cost of grading accuracy as with the clustering interface reaching 90% of grading accuracy across all correct/incorrect answers across the two questions.

Discussion

The clustering interface drastically reduced the grading time while decreasing the reliability of grades. However, it allowed instructors to become part of the grading experience and provide feedback. This is a significant departure from other grading tools and helps attack the tw-way feedback problem: where learners do not receive feedback from instructors and instructors do not receive feedback from learner submissions about their confusion or their problem areas because they never see them!

The success of clustering in reducing instructor effort in grading with its associated grading reliability trade-off is supported by statistical predictions of MOOC assessment [52]. Moreover, for a sufficiently small cluster size, the number of wrongly graded submissions can be reduced while increasing instructor effort.

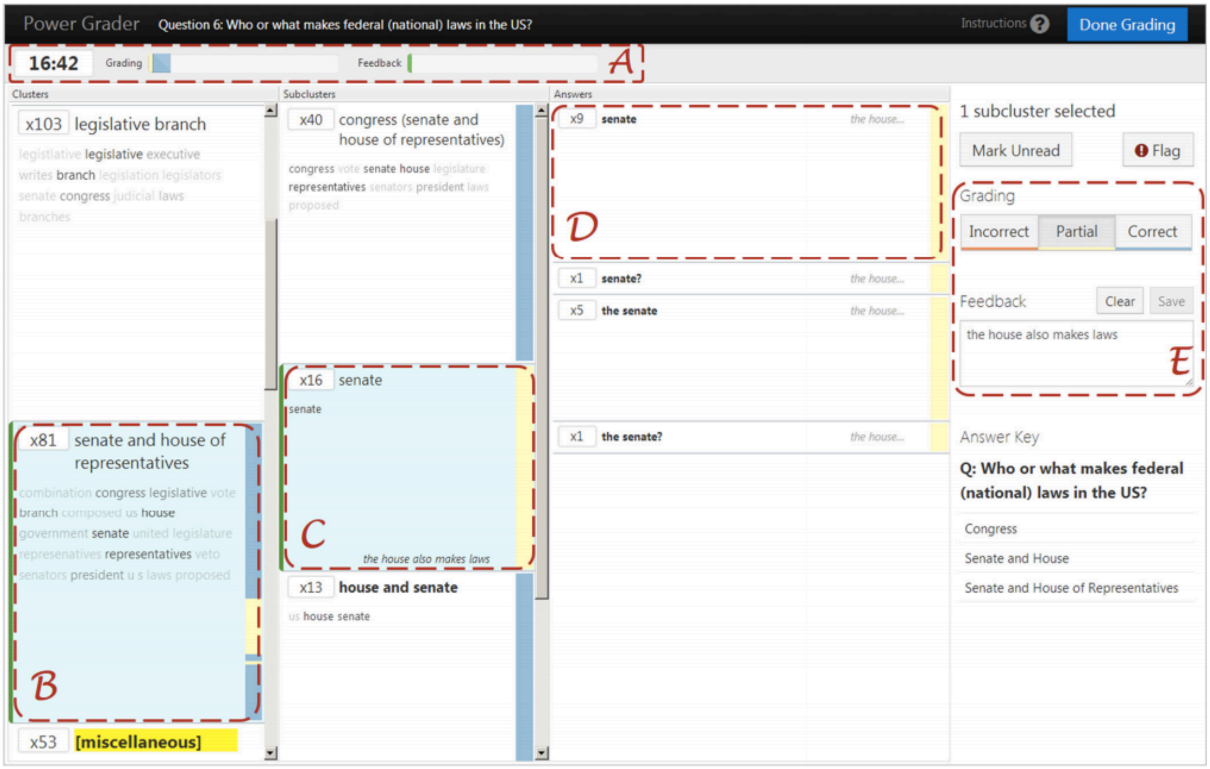

Power Grader
Question 6: Who or what makes federal (national) laws in the US?
Instructions
Done Grading
16:42
Grading
Feedback
A
Clusters
x103 legislative branch
legislative legislative executive writes branch legislation legislators senate congress judicial laws branches
x81 senate and house of representatives
combination congress legislative vote branch composed us house government senate united legislature representatives representatives veto senators president u s laws proposed
B
x53 [miscellaneous]
Subclusters
x40 congress (senate and house of representatives)
congress vote senate house legislature representatives senators president laws proposed
x16 senate
senate
C
the house also makes laws
x13 house and senate
us house senate
Answers
x9 senate
the house...
D
x1 senate?
the house...
x5 the senate
the house...
x1 the senate?
the house...
1 subcluster selected
Mark Unread
Flag
Grading
Incorrect
Partial
Correct
Feedback
Clear
Save
the house also makes laws
E
Answer Key
Q: Who or what makes federal (national) laws in the US?
Congress
Senate and House
Senate and House of Representatives

Figure 8: An interface that shows clustered answers for the instructors to grade and provide feedback, from [11]. Components of interface include: (A) Progress bars, (B) Cluster of 81 learners, graded mostly correct (blue), (C) sub-cluster, graded partially correct (yellow), (D) answer “senate” provided by 9 learners, (E): space to provide grades and feedback

Can clustering help provide feedback on essays and non-writing

artifacts such as design of programs?

Clustering short answers is challenging since they have lesser number of features. However, this also improves the chances of finding right clusters for a submission for a short-answer as compared to an essay. Therefore, relying on clustering for essays will require reviewing Powergrading algorithm that powers the Divide and Cluster work, for essays. Moreover, relying on the right features might break out of the domain of textual responses. For instance, in the case of Mathematical Language Processing [cite MLP work], the domain of possible inputs is large. For example, a learner can represent e^{-x} as 1/{e^x} and the two expressions will lead to different feature vectors. Renaming variables can have similar effects where the feature vectors for equivalent solutions do not put them in the same cluster.

To summarize, "cluster, then instructor" is a useful paradigm to approach feedback at scale. Similar techniques have been attempted for other domains like grading programming assignments. While this approach can provide significant gains in grading time while maintaining accuracy of grades, it is restricted to domains where clustering features are a reasonable approximation of the information and intent content of a submission. Moreover, intelligent ways to reduce instructor work towards finding the right features for clustering could be helpful, along with building useful instructor tools for rapid grading and feedback of clustered submissions.

Similarly, design of a program is a subjective issues with many features, most of which cannot be quantified. E.g.: the choice of variable names, the layers of abstraction, ease of creating APIs etc. Since people use many programming styles, software companies enforce their own standards (https://github.com/google/styleguide) for their software developers using code review practices. However, this is difficult to do for novice programmers since they are focused on low-level details and might not have the skills/time to focus on high-level code design unlike well-trained software engineers at companies. Hence, it's even more important to provide them feedback on their code design. There is no mention of automated tools that provide design feedback apart from trivial code indentation. Since people are better at recognizing high-level features, systems have been developed that use peers or crowds or experts to provide feedback on the design of novice's programs [37].

PEER ASSESSMENT CAN HELP SCALE ESSAY FEEDBACK

Peer assessment can be considered to be two different problems: *peer grading* and *peer feedback.* In peer grading, learners need to grade a submission without having to provide any comments (or minimal comments). Peer grading is concerned with ascertaining the *quality* of the submission on a numeric scale. In peer feedback, learners need to provide subjective comments as a critique intended to offer ways of improvement.

Peer assessment improves reflective abilities and builds learner bonds

Improved understanding via reflection: In collocated environments, viewing and commenting on others' work helps set norms and provides learning gains by iteratively improving their work and engaging in reflection [50]. The process of critiquing someone else's work opens learners to new ideas and ways to look at their own work to improve them. Previous research [14,57] has found that assessing others' work yields perspective and inspiration, while assessing your work after others' yields reflection and comparison [12].

Solving Alone together: Due to exchanging feedback, learners are more likely to feel part of a larger group rather than feeling alone.

Reduced instructor involvement: Peer assessment scales well. By enabling peer assessment, instructors do not pend time assessing every submission, but instead, they need to focus on making the learners better graders.

Peer assessment consumes time and provides delayed feedback of questionable quality

Novice and expert divide: Peers taking a class are not experts in the domain. Expecting them to provide feedback similar to an expert is unrealistic. Peers focus on superficial features while experts highlight deeper, conceptual features [cite - pull citation from CSCW submission]. "For example, novices and experts differ in the details they attend to: novices often overvalue detailed and surface-level concerns and undervalue conceptual and meaning-level issues. Novices may focus on grammar errors to the exclusion of revising logical flow in a writing assignment [67], and struggle to identify global problems [20, 25]. Expert reviewers may provide clearer justifications [19]."

Consumes time: Providing a high-quality review requires time and effort to understand the deeper features of work and comment on them. However, online learners have limited time to spare for their courses and would, arguably, spend lesser time providing reviews to other learners. 75% of Online HCI learners are over 24 years old (Figure 11). Independent studies have found similar trends across other MOOC platforms [30]. These learners are different from traditional university learners in many ways, as shown in Figure 11.

First, learning is a side-activity for online learners while they manage their main responsibilities like job and family. Second, Learners have limited peer interactions online. So, even though there might be peers who would be struggling with the same concepts in an online class, they have limited ways to automatically find them since there's no colocation. This problem has been called as "Alone Together" [60]. Third, unlike university learners taking broad set of classes, online learners have specific motives for doing courses. The objective of many online learners is retooling or developing better skills for their current or next job [30]. This trend is

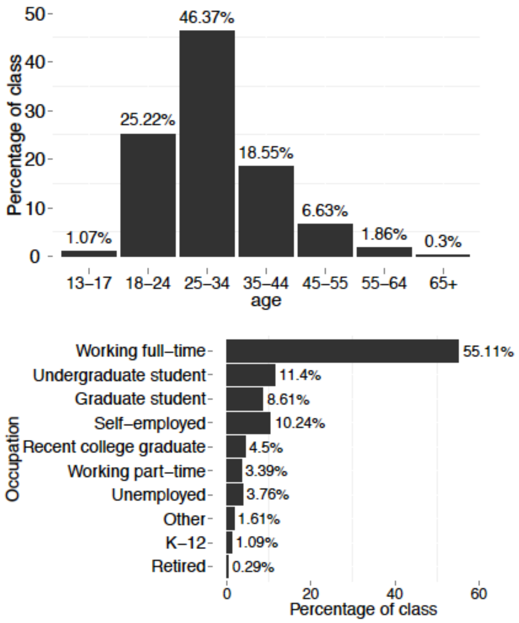


Figure 9: Nearly three-quarters of an online class were 18-34 years old and over half of the learners work full-time. Data from Fall 2012 Online HCI class on Coursera [30]

pervasive across different MOOC platforms and courses. In an independent evaluation, first Database online class found that most of its learners were software professionals who wanted to improve their skills on the job [65].

Delay in feedback: Unlike automated techniques, feedback from learners is not instantaneous and requires delay. This can be unsuitable for some tasks such as programming that requires quickly fixing errors to move on, or even for rapid iteration on essays.

In sections below, I describe systems that tackle the three limitations of peer assessment mentioned above.

Bridging novice and expert divide: peers grade similar to staff but structure of rubrics is important

An evaluation of Coursera peer assessment technique studied the efficacy of peer and self-assessment (where learners rate their own work) by comparing learner-assigned grades to instructor staff grades, and identified key challenges to improving the quality [32]. Via **three experiments in a massive course of 30K learners (check)**, three insights emerged. First, giving learners feedback about their grading bias increased subsequent accuracy. Second, introducing short, customizable feedback snippets, a peer-analogue of error models, that cover common issues with assignments, provided learners more qualitative peer feedback by reducing their effort. Looking into the rubrics used for assessment, high-variance items for improvement were identified. Testing rubrics that use parallel sentence structure, unambiguous wording and wells specified dimensions demonstrated lower variance. After revising rubrics, median grading error decreased from 12.4% to 9.9%." Peer grades correlated with staff-assigned grades, with 42.9% of learners grades within 5% of the staff grade, and 65.5% within 10%, as shown in Figure 12. Interestingly, increasing the number of raters beyond three did not yield sufficient improvements in grading accuracy (~5% grading accuracy increase from 3 to 7 raters), demonstrating that the extra effort on part of every learner does not scale the quality of results.

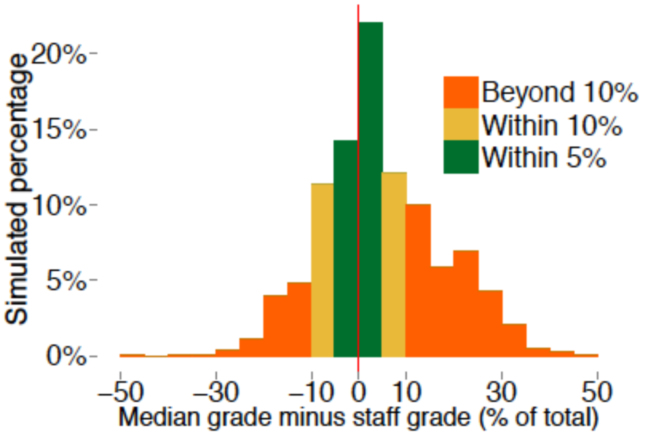


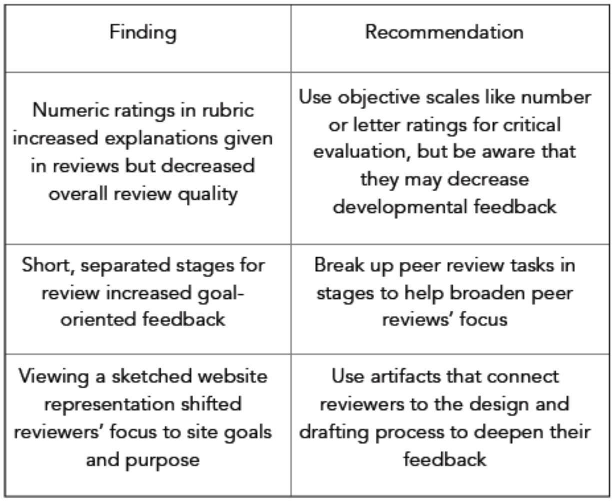
Figure 10: 42% of peer grades within 5% of staff grades, and 65% within 10% [30]

Peer assessment tools amplify instructor inputs as well

In Coursera peer assessment study, rubrics and feedback snippets are instructor inputs that are used by learners to assess others' work. Much like instructors identified features for clustering algorithms, and error models for automated programming feedback tools, they allow the technique to scale instructor knowledge to a large number of learner submissions with decreased reliability. Further research[1] [27] has studied the role of the review environment in framing peer reviews, going beyond rubrics and looking at various environmental features such as presence or absence of numeric ratings, or stage-wise decomposition of peer review and the representation of the draft. Table 2 provides a summary. These could be choices made by instructors along with learning platforms.

[1] Original work with Catherine Hicks and Ailie Fraser (Design Lab, UC San Diego). Accepted at CHI 2016

Table 2: Framing peer review environment to elicit better quality reviews [27]



Finding	Recommendation
Numeric ratings in rubric increased explanations given in reviews but decreased overall review quality	Use objective scales like number or letter ratings for critical evaluation, but be aware that they may decrease developmental feedback
Short, separated stages for review increased goal-oriented feedback	Break up peer review tasks in stages to help broaden peer reviews' focus
Viewing a sketched website representation shifted reviewers' focus to site goals and purpose	Use artifacts that connect reviewers to the design and drafting process to deepen their feedback

Reducing effort: Learners provide relative ranks to submissions rather than assigning explicit scores

An alternate statistical approach to solve the inconsistency of peer grading that can also reduce learner effort includes using ordinal scores (ranking) rather than cardinal (absolute scores) to. People are better at providing relative order of items than ascertaining their absolute value. In cardinal grading, peers provide explicit grades to the submissions they see, but in ordinal scores, peers provide an order for the quality of submissions provided to them. Since the objective of large-scale assessment is to assign scores to all the submissions, rank-aggregation schemes can be used to take these partial ranks and create a total rank. Inferring the total order from the partial order over a subset of submissions is called the **ordinal peer grading** problem [64].

BayesRank [47] and Bayesian Ordinal Peer Grading [64] tackle the ordinal peer grading problem. However, going beyond the naïve approach of providing just the order [51], Bayesian ordering also provides the confidence in every ranking to help instructors understand how to score these assignments. Bayesian ordinal grading studies the following problems: a) the quality of its predicted rankings in comparison with existing peer-grading methods as measured with regards to conventional instructor grades; and b) the accuracy of the confidence intervals and uncertainty information. The performance of the proposed Bayesian methods is not substantially different from that of the standard techniques. This implies that Bayesian ordinal grading is equivalent in its correctness to standard techniques while also providing the extra information about the confidence of ratings **(add data)**.

Providing rapid feedback: Peers cannot compete with automated tools in providing rapid feedback

Peerstudio explored how rapid peer feedback can reduce the time needed to receive feedback on an essay draft. To incentivize learners to provide quick feedback, they could receive feedback on their submissions only when they provided rubric-based feedback to two peers' drafts. Interestingly, learners could use the system as many times as they desired to continuously improve their essay. This is line with mastery learning principles where learners must repeatedly revise based on immediate, focused feedback [18]. Rapid, corrective feedback improves learners revise their work and in writing, programming and art, rapid feedback has demonstrated reliably improving learning [31].

In a MOOC where 472 learners used Peerstudio, reviewers were recruited within minutes (median wait time: seven minutes), and the first feedback was completed soon after (median wait time: 20 minutes). Learners in the two, smaller, in-person classes received feedback in about an hour on average. Rapid feedback on in-progress work improves course outcomes: in a controlled experiment, learners' final grades improved when feedback was delivered quickly, but not if delayed by 24 hours.

OPPORTUNITY: COMBINE AUTOMATED TOOLS AND PEOPLE (INSTRUCTORS AND PEERS)

While peer assessment can be substantially improved by seeing it with instructor artifacts like rubrics, feedback snippets and using constraints to incentivize participation, unsurprisingly it does not compare to the consistency and latency of automated grading. However, peers and instructors can provide open-ended subjective feedback specific to the learner submission, which is beyond the scope of current tools for writing tasks. This is in line with known ideas that machines are better suited to repetitive tasks that can be precisely described in a sequence of steps. Identifying creative solutions is not the forte of current algorithms. People are far superior at handling exceptional cases. How can people and machines be combined to provide useful feedback to learners?

Combining human expertise to help machines solve tasks they are inherently poor at has been put to use in many topics [1]. In the context of online classes, entirely instructor-graded assignment does not scale for thousand of learners and completely machine-graded submissions do not provide good results. Peer grading problem lends itself to be attempted with a combination of the two approaches where peers and machines can collaboratively grade some submissions.

$$\begin{aligned}
\text{(Reliability) } & \tau_v \sim \mathcal{G}(\alpha_0, \beta_0) \text{ for every grader } v, \\
\text{(Bias) } & b_v \sim \mathcal{N}(0, 1/\eta_0) \text{ for every grader } v, \\
\text{(True score) } & s_u \sim \mathcal{N}(\mu_0, 1/\gamma_0) \text{ for every user } u, \text{ and} \\
\text{(Observed score) } & z_u^v \sim \mathcal{N}(s_u + b_v, 1/\tau_v), \\
& \text{for every observed peer grade.}
\end{aligned}$$

Figure 13: Models for grader reliability, bias, true score and observed score. Grader reliability is assumed to be a Gaussian to model inherent learner ability to grade while others are assumed to be normal variables, that is affected by external factors. The parameters are learned from the data.

Example System: Grade with peers but weigh grades based on grader history

While peer grading scales well, it does not provide the same quality as expert grades. Can peer grader biases be modeled and used to suitably weigh grades provided by them. For instance, if a learner's grades correlate well with median grade, then it can be weighed more than another learner's grade whose grading shows consistent different from the median grade. Can estimating and correcting for grader biases and reliabilities significantly improve peer grading accuracy?

Figure 13 shows the probabilistic model built for peer grading at scale in [43] using the following inputs: (a) true score (Latent variable that represents the *true underlying score* for a submission, which needs to be estimated), (b) Grader bias: Variable that represents a grader's tendency to inflate/deflate his/her assessment, (c) Grader reliability: Variable that models how well does a grader's assigned score corresponds to the true score, (d) Observed grades: Represents the actual scores provided by peer graders

Simple model with Grader bias and reliability: Use prior distributions over latent variables. The key assumption is that an individual grader's bias may be nonzero, but the average bias of many graders is zero.

Modeling grader biased yielded big benefits. However, modeling grader precision provided small benefits. Runs on synthetic dataset showed that modeling variance provides value when the number of assignments being graded is more than 10. This does not set in well with existing schemes where learners provide grades to 4 or 5 other learners.

One important question in this model is about the question of existence of a true score for a particular submission, and also, about the peer median grade being aa the true reflection of that. While wisdom of the crowds [56] is known to perform better than a single non-expert's answer, it is not considered to be synonymous to the true grade.

Example System: Grade algorithmically but verify grades with peers

While peers can grade open-ended submissions using rubrics [32], it takes a large amount of time. Can the grading burden be lowered while maintaining quality for **short-answer questions**? Unlike open-ended assignments like essays where algorithmic grading fails to develop a semantic understanding, short-answer questions have lesser deeper features that might allow a better estimate of the quality of the submission [33]. This insight is used towards developing a solution that combines grading algorithms with verification using peers. This preserves robustness provided by peer grading while lowering their burden.

The paper introduces the identify-verify pattern, a grading algorithm first predicts a learner grade and estimates confidence, which is used to estimate the number of peer raters required. Peers then identify key features of the answer using a rubric. Finally, other peers verify whether these feature labels were accurately applied. This pattern adjusts the number of peers that evaluate an answer based on algorithmic confidence and peer agreement.

The algorithm works through four main stages as shown in Figure 14. In Stage 1, grading algorithms predict a grade for the submission along with a confidence value. The confidence value determines the number of peers needed for verification of the grade. In Stage 2 (Identify phase), peers use rubric to identify the presence/absence of attributes in the answer. In Stage 3 (Verify phase), two other peers verify that such attributes do exist. The final score is generated as the sum of verified attributes. If an identifier rejects the attributes, the task is put back for identification one more time. For two independent identifications being identical, one is considered verification.

The identify-verify pattern was tested with 1370 learners from an online human-computer interaction class. The manipulation included the different systems: peer median (existing choice in peer grading platforms), Identify only and Identify Verify pattern. Two metrics were used for correctness and efficiency of the new pattern. For correctness of peer grading, for each learner answer, the staff grade was compared to the computed grade. For efficiency, median time was computed for time taken to

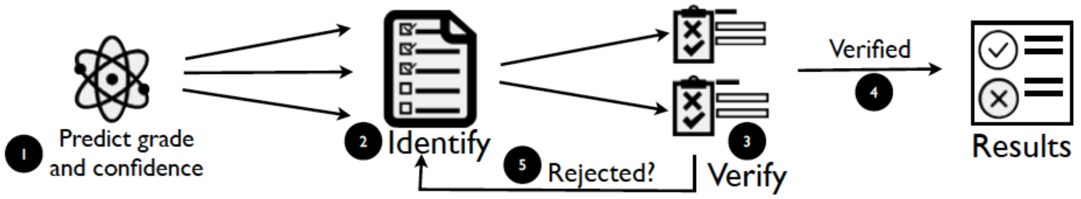


Figure 14: Algorithms predict grade and confidence while peers verify the grade. (1) Algorithm predicts grades and confidence which decides number of independent identifications (2) Peers select answer attributes that matches (3) Two other peers verify Step 2 (4) Final score is sum of verified attributes (5) If one or more peers reject attributes, another peer is asked to rate. If two or more peers have identical results, one is assumed to be a verifier [31]

grade a question.

OPPORTUNITY: REDUCE INSTRUCTOR EFFORT AND QUANTIFY LEARNER IMPROVEMENTS

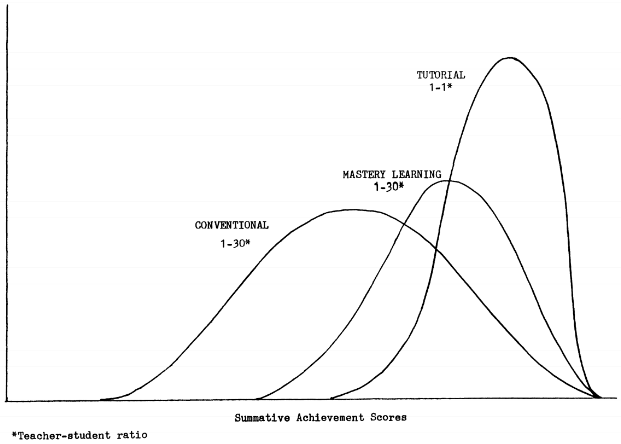


Figure 11: Achievement distribution under different learning paradigms: conventional, mastery and tutorial instruction, from [10]. Tutoring shows great benefits in peak performance as well as average performance of distribution

Useful instructor inputs are critical to the performance of automated tools. Hence, instructor resources need to be utilized optimally. A user-centered design process for the instructor would help system-builders understand their challenges. This might show inefficiency in interacting with student submissions or in the actual process of providing feedback. For example, common feedback elements could be used as templates to allow instructors to use their time wisely. Such challenges and solutions will emerge from a study of instructors working with a large scale of learners.

As Mentioned in Bloom's seminal work [10], 1:1 tutoring is the most reliable way to improve learner performance. The principles and opportunities mentioned in this synthesis are intended to inform system-design for providing better feedback at scale. By providing useful feedback to learners on their creative assignments, we can get a step closer to the holy grail of customized tutoring for online learners depending on their unique learning pace and environment.

ACKNOWLEDGMENTS

I thank my advisor Scott Klemmer for pointing out relevant research and questioning my synthesis till it made sense. I thank all members of the Design Lab for tolerating me day in and out.

REFERENCES

1. von Ahn, L., Maurer, B., McMillen, C., Abraham, D., & Blum, M. ReCAPTCHA: human-based character recognition via web security measures. Science, 321:5895, (Sept. 12, 2008), 1465-1468.
2. R. Alur, L. D'Antoni, S. Gulwani, D. Kini, and M. Viswanathan. Automated Grading of DFA Constructions. In Proceedings of the Twenty-Third International Joint Conference on Artificial Intelligence, IJCAI'13, pages 1976–1982. AAAI Press, 2013.
3. Paolo Antonucci, Christian Estler, Durica Nikolić, Marco Piccioni, and Bertrand Meyer. 2015. An Incremental Hint System For Automated Programming Assignments. In *Proceedings of the 2015 ACM Conference on Innovation and Technology in Computer Science Education* (ITiCSE '15). ACM, New York, NY, USA, 320-325. DOI=http://dx.doi.org/10.1145/2729094.2742607
4. Basu, S., Jacobs, C., and Vanderwende, L. Powergrading: a Clustering Approach to Amplify Human Effort for Short Answer Grading. TACL 1, (2013), 391–402.
5. R.E. Bennett, M. Steffen, M.K. Singley, M. Morley, and D. Jacquemin. 1997. Evaluating an Automatically Scorable, Open-Ended Response Type for Measuring Mathematical Reasoning in Computer-Adaptive Tests. Journal of Educational Measurement 34, 2 (1997), 162–76.
6. Michael S. Bernstein, Greg Little, Robert C. Miller, Björn Hartmann, Mark S. Ackerman, David R. Karger, David Crowell, and Katrina Panovich. 2015. Soylent: a word processor with a crowd inside.*Commun. ACM* 58, 8 (July 2015), 85-94. DOI=10.1145/2791285 http://doi.acm.org/10.1145/2791285
7. Multiple-choice questions. Bishop.
8. Kyle Bishop. Essay grading rubric. https://www.suu.edu/faculty/bishopk/pdf/essay-rubric.pdf
9. Bloom, B.S., et al. Taxonomy of Educational Objectives. Handbook I: Cognitive C main. New York. Mc ay. 1956.
10. Bloom, B. S. (1984) The 2 Sigma Problem: The search for methods of instruction as effective as one-to-one tutoring. Educational Leadership, 41(8), 4-17.
11. Michael Brooks, Sumit Basu, Charles Jacobs, and Lucy Vanderwende. 2014. Divide and correct: using clusters to grade short answers at scale. In Proceedings of the first ACM conference on Learning @ scale conference (L@S '14). ACM, New York, NY, USA, 89-98. DOI=http://dx.doi.org/10.1145/2556325.2566243
12. D. Boud. 1995. Enhancing learning through self-assessment. Routledge. **1994 or 1995**
13. Julia Cambre, Chinmay Kulkarni, Michael S. Bernstein, and Scott R. Klemmer. 2014. Talkabout: small-group discussions in massive global classes. In *Proceedings of the first ACM conference on Learning @ scale conference* (L@S '14). ACM, New York, NY, USA, 161-162. DOI=http://dx.doi.org/10.1145/2556325.2567859
14. D. Chinn. 2005. Peer assessment in the algorithms course. ACM SIGCSE Bulletin 37, 3 (2005), 69–73
15. Life after MOOCs. http://cacm.acm.org/magazines/2015/10/192385-life-after-moocs/fulltext **(FIX THIS CITATION)**

16. L. D'antoni, D. Kini, R. Alur, S. Gulwani, M. Viswanathan, and B. Hartmann. How can automatic feedback help learners construct automata&quest. ACM Transactions on Computer-Human Interaction (TOCHI), 22(2):9, 2015.
17. Diversity in MOOC learners' backgrounds and behaviors in relationship to performance in 6.002 x J DeBoer, GS Stump, D Seaton, L Breslow Proceedings of the Sixth Learning International Networks Consortium Conference
18. Ericsson, K.A., Krampe, R.T., and Tesch-Römer, C. The role of deliberate practice in the acquisition of expert performance. Psychological review 100, 3 (1993).
19. Alvin Yuan, Kurt Luther, Markus Krause, Sophie Vennix, Steven P. Dow, Björn Hartmann. "Almost an Expert: The Effects of Rubrics and Expertise on Perceived Value of Crowdsourced Design Critiques." To appear in Proceedings of CSCW 2016.
20. Nancy Falchikov and Judy Goldfinch. 2000. Learner peer assessment in higher education: A meta-analysis comparing peer and teacher marks. Rev of Ed Res. 70, 3: 287-322.
21. E. Fast, C. Lee, A. Aiken, M. S. Bernstein, D. Koller, and E. Smith. Crowd-scale Interactive Formal Reasoning and Analytics. In Proceedings of the 26th Annual ACM Symposium on User Interface Software and Technology, UIST '13, pages 363–372, New York, NY, USA, 2013. ACM.
22. John Hattie and Helen Timperley. 2007. The Power of Feedback. Review of Educational Research 77, 1 (March 2007), 81–112. DOI: http://dx.doi.org/10.3102/003465430298487
23. Geiger, M.A. & Simons, K.A.(1994). Intertopical sequencing of multiple-choice questions: Effect on exam performance and testing time. Journal of Education for Business, 70(2), 87-90.
24. Sumit Gulwani, Ivan Radiček, and Florian Zuleger. 2014. Feedback generation for performance problems in introductory programming assignments. In Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering (FSE 2014). ACM, New York, NY, USA, 41-51. DOI=http://dx.doi.org/10.1145/2635868.2635912
25. John R. Hayes, Linda Flower, Karen A. Schriver, James F. Stratman, and Linda Carey. 1987. Cognitive processes in revision. In Advances in applied psycholinguistics, Volume II: Reading, writing, and language processing, Sheldon Rosenberg (Ed.). Cambridge, England, Cambridge University Press, 176–240.
26. Hearst, M. (2000). The debate on automated essay grading. Intelligence Systems 15.5, 22–37.
27. Framing Feedback: Choosing Review Environment Features that Support High Quality Peer Assessment. To be presented at CHI 2016.
28. G. Juniwal, A. Donz´e, J. C. Jensen, and S. A. Seshia, "CPSGrader: Synthesizing temporal logic testers for auto-grading an embedded systems laboratory," in Proceedings of the 14th International Conference on Embedded Software (EMSOFT), October 2014.
29. Retrieval Practice Produces More Learning than Elaborative Studying with Concept Mapping. Jeffrey D. Karpicke* and Janell R. Blunt. Science 2012.
30. https://www.insidehighered.com/news/2012/06/05/early-demographic-data-hints-what-type-learner-takes-mooc
31. Kulik, J.A. and Kulik, C.-L.C. Timing of Feedback and Verbal Learning. Review of Educational Research 58, 1 (1987), 79– 97.
32. Chinmay Kulkarni, Koh Pang Wei, Huy Le, Daniel Chia, Kathryn Papadopoulos, Justin Cheng, Daphne Koller, Scott R. Klemmer, 2013. Peer and Self Assessment in Massive Online Classes. ACM Trans. Comput.- Hum. Interact. 9, 4, Article 39 (March 2013), 31 pages. DOI:http://dx.doi.org/10.1145/0000000.0000000
33. Chinmay E. Kulkarni, Richard Socher, Michael S. Bernstein, and Scott R. Klemmer. 2014. Scaling short-answer grading by combining peer assessment with algorithmic scoring. In Proceedings of the first ACM conference on Learning @ scale conference (L@S '14). ACM, New York, NY, USA, 99-108. DOI=http://dx.doi.org/10.1145/2556325.2566238
34. Chinmay E. Kulkarni, Michael S. Bernstein, and Scott R. Klemmer. 2015. PeerStudio: Rapid Peer Feedback Emphasizes Revision and Improves Performance. In Proceedings of the Second (2015) ACM Conference on Learning @ Scale (L@S '15). ACM, New York, NY, USA, 75-84. DOI=http://dx.doi.org/10.1145/2724660.2724670
35. Landauer, T. K., Foltz, P. W., & Laham, D. (1998). Introduction to Latent Semantic Analysis. Discourse Processes, 25, 259-284.
36. Derek Lomas, Kishan Patel, Jodi L. Forlizzi, and Kenneth R. Koedinger. 2013. Optimizing challenge in an educational game using large-scale design experiments. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (CHI '13). ACM, New York, NY, USA, 89-98. DOI=http://dx.doi.org/10.1145/2470654.2470668
37. Medium. How a Udacity graduate earns $11k a month reviewing code. Retrieved September 21, 2014 from http://www.medium.com/@olivercameron/how-audacity- graduate-earns-11k-a-month-reviewing-codec2a7d295724c
38. Joseph Bahman Moghadam, Rohan Roy Choudhury, HeZheng Yin, and Armando Fox. 2015. AutoStyle: Toward Coding Style Feedback at Scale. In *Proceedings of the Second (2015) ACM Conference on Learning @ Scale* (L@S '15). ACM, New York, NY, USA, 261-266. DOI=http://dx.doi.org/10.1145/2724660.2728672

39. Nesterko, S. O., Seaton, D. T., Kashin, K., Han, Q., Reich, J., Waldo, J., Chuang I., & Ho, A. D. (2014). World Map of Enrollment (HarvardX Insights).
40. Donald A. Norman. 2002. *The Design of Everyday Things*. Basic Books, Inc., New York, NY, USA.
41. Eleanor O'Rourke, Christy Ballweber, and Zoran Popovií. 2014. Hint systems may negatively impact performance in educational games. In Proceedings of the first ACM conference on Learning @ scale conference (L@S '14). ACM, New York, NY, USA, 51-60. DOI=http://dx.doi.org/10.1145/2556325.2566248
42. Perelman, D., Gulwani, S. & Grossman, D. (2014). Test-Driven Synthesis for Automated Feedback for Introductory Computer Science Assignments. In Data Mining for Educational Assessment and Feedback (ASSESS 2014).
43. Tuning Peer Grading C. Piech, J. Huang, Z. Chen, C. Do, A. Ng, D. Koller Proceedings of the 6th International Conference on Educational Data Mining, Memphis, USA. 2013
44. Chris Piech, Mehran Sahami, Jonathan Huang, and Leonidas Guibas. 2015. Autonomously Generating Hints by Inferring Problem Solving Policies. In *Proceedings of the Second (2015) ACM Conference on Learning @ Scale* (L@S '15). ACM, New York, NY, USA, 195-204. DOI=http://dx.doi.org/10.1145/2724660.2724668
45. Michael Prince. 2004. Does active learning work? A review of the research. Jour of Eng Educ. 93: 223-232.
46. Chris Quintana, Brian J. Reiser, Elizabeth A. Davis, Joseph Krajcik, Eric Fretz, Ravit Golan Duncan, and Elliot Soloway. (2004). A scaffolding design framework for software to support science inquiry. The Jour of the Learn Sci. 13, 3: 337-386.
47. Karthik Raman and Thorsten Joachims. 2015. Bayesian Ordinal Peer Grading. In *Proceedings of the Second (2015) ACM Conference on Learning @ Scale* (L@S '15). ACM, New York, NY, USA, 149-156. DOI=http://dx.doi.org/10.1145/2724660.2724678
48. Rivers, K., and Koedinger, K. R. Automating hint generation with solution space path construction. In Intelligent Tutoring Systems, Springer (2014), 329–339.
49. D. Royce Sadler. 1989. Formative assessment and the design of instructional systems. Instructional Science 18, 2 (June 1989), 119–144. DOI: http://dx.doi.org/10.1007/BF00117714
50. D. Schon. 1985. The Design Studio: An exploration of its traditions and potential. ¨ London: Royal Institute of British Architects (1985).
51. N. B. Shah, J. K. Bradley, A. Parekh, M. Wainwright, and K. Ramchandran. A case for ordinal peer-evaluation in MOOCs. In NIPS Workshop on Data Driven Education, Dec. 2013.
52. N. B. Shah, J. Bradley, S. Balakrishnan, A. Parekh, K. Ramchandran, and M. J. Wainwright, "Some scaling laws for MOOC assessments," in KDD Workshop on Data Mining for Educational Assessment and Feedback (ASSESS 2014), 2014.
53. R. Singh, S. Gulwani, and S. Rajamani. Automatically generating algebra problems. In AAAI, 2012
54. Rishabh Singh, Sumit Gulwani , Armando Solar-Lezama, Automated feedback generation for introductory programming assignments, Proceedings of the 34th ACM SIGPLAN conference on Programming language design and implementation, June 16-19, 2013, Seattle, Washington, USA
55. Nancy Sommers and Laura Saltz. The Novice as Expert: Writing the Freshman Year College Composition and Communication. Vol. 56, No. 1 (Sep., 2004), pp. 124-149
56. James Surowiecki. 2005. *The Wisdom of Crowds*. Anchor.
57. D. Tinapple, L. Olson, and John Sadauskas. 2013. CritViz: Web-Based Software Supporting Peer Critique in Large Creative Classrooms. Bulletin of the IEEE Technical Committee on Learning Technology 15, 1 (2013), 29.
58. [Thompson2000] ??? – chinmay coursera work
59. Chih-Hsiung Tu and Marina McIsaac. 2002. The relationship of social presence and interaction in online classes. The Am Jour of Dis Educ. 16, 3: 131-150.
60. Turkle, S. Alone Together: Why We Expect More from Technology and Less from Each Other. Basic Books, Inc., (2011).
61. Kurt VanLehn. 2006. The behavior of tutoring systems. International Journal of Artificial Intelligence in Education 16, 3, 227–265. http://dl.acm.org/citation.cfm?id=1435351.1435353
62. Veloski et al (1999) Patients don't present with five choices: an alternative to multiple-choice tests in assessing physicians' competence.
63. Joe Warren, Scott Rixner, John Greiner, and Stephen Wong. 2014. Facilitating human interaction in an online programming course. In Proceedings of the 45th ACM technical symposium on Computer science education (SIGCSE '14). ACM, New York, NY, USA, 665-670. DOI=http://dx.doi.org/10.1145/2538862.2538893
64. Andrew E. Waters, David Tinapple, and Richard G. Baraniuk. 2015. BayesRank: A Bayesian Approach to Ranked Peer Grading. In *Proceedings of the Second (2015) ACM Conference on Learning @ Scale* (L@S '15). ACM, New York, NY, USA, 177-183. DOI=http://dx.doi.org/10.1145/2724660.2724672
65. J. Widom. 2012. From 100 Learners to 100,000. ACM SIGMOD Blog (http://wp.sigmod.org/?p=165).
66. Winerip, M. Facing a robo-grader? just keep obfuscating mellifluously. New York Times (2013).
67. Vivian Zamel. 1985. Responding to learner writing. Tesol Quarterly. 19, 1: 79-101

APPENDIX: TAXONOMY OF FEEDBACK

Feedback depends on learning objectives and nature of learner mistakes

The type of feedback for a certain assignment is tied to the learning objective. In domains where the objective is to get the right answer (such as solving a basic numerical to calculate area of a circle), feedback can be provided as hints to fix the wrong portions of the solution. In creative domains with multiple successful solutions (such as programming puzzles or math proofs or essays), high-level feedback and broad suggestions on the ideas and techniques used can help learners reflect on their work and consider a broader design space. If they are in implementation phase, providing quick fixes can help learners get unstuck on low-level details. Feedback can also be aimed at either refreshing learner's knowledge or towards teaching them a new technique. Multiple solutions/ examples can be presented as a form of feedback to increase the space of exploration of the learner. Hence, depending on the learning objective, feedback can focus on the global structure of the solution or it might suggest local changes to fix wrong or missing details. Typically, the instructor will decide these learning objectives.

When feedback is meant to provide guidance around fixing a broken solution, it can take two types, depending on the mistakes [39]. If the mistake is at the planning stage called an error, then the feedback needs to be at an abstract level pointing out why the current plan fails. For example, if a learner needs to implement a sorting algorithm but instead uses a binary search, in this case the feedback needs to make the learner aware of their error in choosing a search algorithm in place of sorting. If the mistake occurs at the execution stage, called a slip, then the feedback should be specific and should clearly point out which step of the work did the learner falter at. These slips might be due to careless mistakes (e.g.: 3 + 2 = 6) or incorrect fast recall.

Feedback has multiple features and forms

From my reading of literature on feedback as well as seeing the nature of feedback offered by automated tools and people, feedback has following important features that can be answering the related questions:

Frequency: How often should the learner receive feedback? Is it whenever the learner submits a solution or makes an important change? Rapid feedback can also help in achieving mastery learning by tackling misconceptions when they first happen, but they can also narrow exploration of alternate solution paths by the learner.

Latency: How quick do you provide feedback when you do?

Coverage: Is the feedback global or local, i.e. does the feedback cover the entire solution or just the locations where there is an error or recent changes? For example, for programs, this relates to entire program level feedback or just line level? More broadly, should the feedback consider the entire space of solutions and provide suggestions of alternate solutions?

Personalization: Is feedback tailored ot the submission or does it reflect general rules? More broadly, should the feedback include learner's preferences i.e. Given the same unsuccessful solution, should the feedback be same across all learners?

Two-way accuracy: Feedback should not provide false alarms: neither false positives nor false negatives.

Distance from a successful solution: A feedback should clarify to the learner which level of wrongness is being talked about.

Cognitive load: How much extra information should the feedback contain or how much extra work should it invoke in a learner? Is the feedback supposed to trigger reflection or it is meant to be actionable?

Three general strategies/forms exist for presenting feedback [61]:

1. Binary feedback that indicates whether a learner's solution is correct or not
2. Counterexamples: Error-specific feedback that points out what is wrong
3. Hints: Solution-oriented feedback that suggests strategies for addressing an error. Some hints might be problem-oriented as well, look below.

Some useful feedback strategies might also involve restating the problem in a different way. For example, many math problems explore symmetric nature of terms. For example, consider solving a basic trigonometry problem. An instructor hint around looking for symmetric terms can help solve the solution better than fixing a long-drawn solution. Feedback can also take the form of successful or unsuccessful examples independent of the learner submission. Many novice programmers do not understand what is really going on inside their program. To fix this lack of transparency, a number of tutors have been developed over the years to visually describe the state of the program. Overall, feedback can take multiple forms depending on the domain, the nature of learner mistake and the instructor goals.

Feedback can have multiple effects

Measuring the utility of feedback follows three approaches.

Direct effect: Improvement in the quality of artifact can provide a measure of utility of feedback. For instance, did the feedback help the learner fix the error in their code or the structure of their essay to create a correct/ successful solution? However, such measures can suffer from the effect of confound variables, such as the ability of the learner to parse and use the feedback received. Moreover, if a feedback reminded a learner of another issue in the code and fixing it fixes the program, then should the feedback be considered successful? Over a long period, does a particular feedback technique lead to better grades or scores.

Indirect effect: Does the feedback lead to a change in learner approach to tasks/ problems? Such utility can be measured by analyzing the nature of errors in learner submissions over time, and also by asking learners about it in surveys.

Equivalent effect: Feedback can be measured by comparing it to the feedback provided by instructors for similar scenarios.