AI2T: Building Trustable AI Tutors by Interactively Teaching a Self-Aware Learning Agent

Anonymous Author(s)

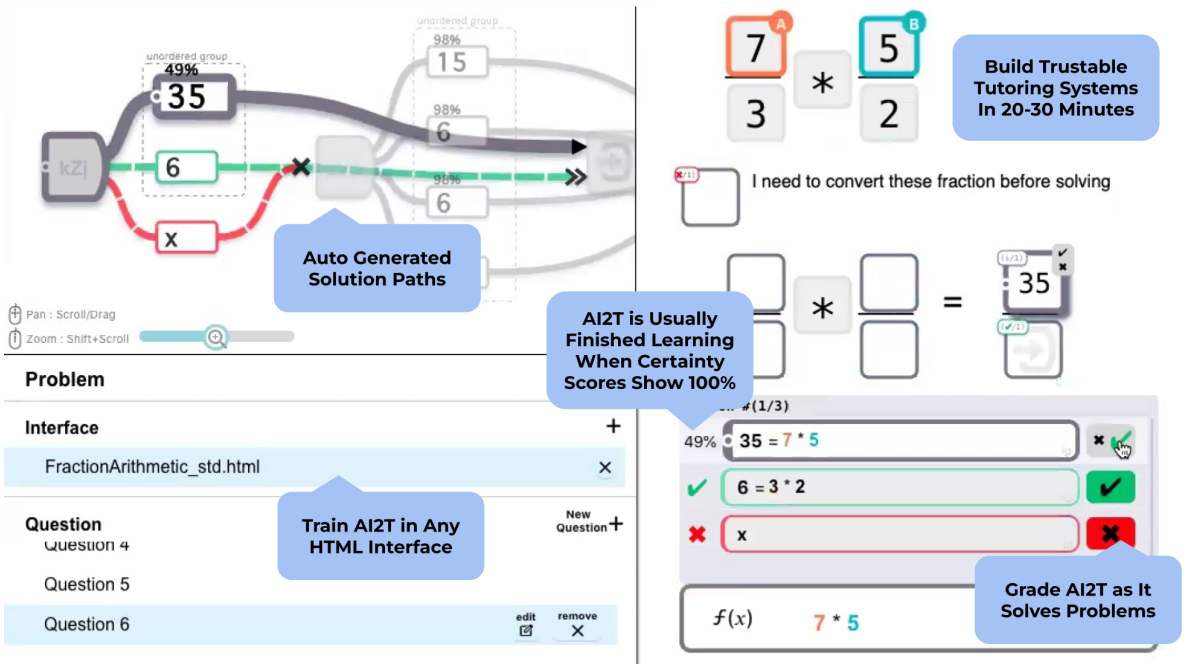


Figure 1: Authors tutor AI2T in HTML interfaces and grade its step-by-step solutions. Certainty scores help authors determine when AI2T has learned robust tutoring programs.

Abstract

AI2T is an interactively teachable AI for authoring intelligent tutoring systems (ITSs). Authors tutor AI2T by providing a few step-by-step solutions and then grading AI2T's own problem-solving attempts. From just 20-30 minutes of interactive training, AI2T can induce robust rules for step-by-step solution tracking (i.e., model-tracing). As AI2T learns it can accurately estimate its certainty of performing correctly on unseen problem steps using STAND: a self-aware precondition learning algorithm that outperforms state-of-the-art methods like XGBoost. Our user study shows that authors can use STAND's certainty heuristic to estimate when AI2T has been trained on enough diverse problems to induce correct and complete model-tracing programs. AI2T-induced programs are more reliable than hallucination-prone LLMs and prior authoring-by-tutoring approaches. With its self-aware induction of hierarchical rules, AI2T offers a path toward trustable data-efficient authoring-by-tutoring for complex ITSs that normally require as many as 200-300 hours of programming per hour of instruction.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
UIST, September 28–October 1st, 2025, Busan, Korea
© 2025 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 978-x-xxxx-xxxx-x/YY/MM
https://doi.org/10.1145/nnnnnnn.nnnnnnn

CCS Concepts

• **Human-centered computing** → **User studies**; **User interface design**; • **Theory of computation** → *Interactive computation*; **Inductive inference**; *Models of learning*; *Active learning*; • **Computing methodologies** → **Learning from demonstrations**; **Online learning settings**; **Rule learning**; *Ensemble methods*; • **Applied computing** → **E-learning**.

Keywords

Interactive Task Learning, Machine Teaching, Interactive Machine Learning, Programming by Demonstration, Intelligent Tutoring Systems, Self-Aware Learning

ACM Reference Format:
Anonymous Author(s). 2025. AI2T: Building Trustable AI Tutors by Interactively Teaching a Self-Aware Learning Agent. In *Proceedings of User Interface Software and Technology (UIST).* ACM, New York, NY, USA, 16 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 Introduction

A dominant theme of the last decade of AI research has been to use big-data machine learning to replicate patterns of behavior distributed across large datasets or explore uses for pretrained models that have been produced in a data-driven manner. By contrast, AI2T (pronounced *A.I. tutee*) is an AI agent that can be **T**aught interactively, in a process that induces **T**rustable well-defined educational programs. These are the two **T**s of AI2T. A central issue with trusting many AI capabilities trained with machine learning is that their learned behaviors are rarely errorless or internally inspectable. Popular methods like deep-learning [37] fit high-dimensional neural networks with inscrutable 'blackbox' weights that can produce very flexible, but often inconsistent behaviors. AI2T avoids these particular issues of trust by inducing well-defined programs expressed as hierarchical task networks (HTNs) [28]. AI2T induces executable symbolic generalizations that are consistent with the author's training.

One way to trust machine-synthesized programs is to inspect the induced programs directly. However, successful code checking requires some programming proficiency and a familiarity with the representation language targeted by the method of program synthesis. AI2T attempts to guide users toward building trustable programs in a different manner. It employs a machine learning method called STAND that can learn efficiently from very little data, yet still accurately estimate its prediction certainty on unseen examples [66]. When STAND's certainty scores tend to increase its true holdout set performance tends to increase. This allows AI2T to be self-aware of its learning, and users can use STAND's certainty scores as an indicator of when AI2T's induced program is complete and trustable. We show in simulation that changes in STAND's certainty estimates more accurately reflect actual changes in holdout set performance than competing predictions from more conventional (and relatively data-efficient) ensemble methods like random forests and XGBoost, which happen to only be as good as chance in this regard. Additionally, we show in a user study that users can successfully use STAND's certainty estimates, as a heuristic for deciding when AI2T has been trained on sufficient practice problems. STAND enables AI2T to be essentially self-aware of its learning progress. This helps users estimate when they have provided AI2T with enough training examples to induce programs with 100% correct behavior.

The core features of AI2T span several machine learning paradigms including machine teaching [69], interactive machine learning, [19], and interactive task learning (ITL) [34]. At its core AI2T builds upon the programming-by-demonstration (PBD) paradigm, where non-programmers demonstrate program behavior instead of writing code in a programming language [18]. Many demonstrations of PBD have shown robust performance in automating simple single action or sequential behaviors [21, 35, 40] that require minimal generalizations from users' demonstrations. However, the challenges of PBD multiply when applied to building larger multi-faceted applications [46, 51]. AI2T can very robustly induce the core behaviors of complex applications known as Intelligent Tutoring Systems (ITS)—educational technologies known for their comprehensive adaptive student support features. Authors train AI2T with a set of interactions that go beyond PBD. These interactions are better described as authoring-by-tutoring [45]—authoring with AI2T involves both demonstrating solutions and interactively checking AI2T's behavior as it attempts to solve problems on its own. In this relationship, the author is the tutor, and AI2T is the tutee. Therefore AI2T is best described by Laird et. al.'s vision of interactive task learning (ITL) [34], the idea of AI systems that can be taught complex and robust new capabilities using a variety of natural interactions that are intuitive to non-programmers.

1.1 Intelligent Tutoring Systems: The Original AI Tutors

Long before recent efforts tutor with generative AI, hand-programmed expert-system-like AI were used to build Intelligent Tutoring Systems (ITS). Decades of learning science research has honed a set of best practices for designing these conventional ITSs [20, 62]. ITSs have been shown to be more effective than traditional classroom instruction alone [62], and in some cases more effective than traditional human-to-human tutoring [33]. The key to historical ITS successes has been instruction designed around learning-by-doing exercises where the ITS provides step-by-step cognitive assistance that adaptively supports students' directed practice [13, 32].

For instance, model-tracing tutors track students' step-by-step solutions to determine in what ways their current knowledge aligns with or diverges from a model of expert knowledge [30, 52]. Model-tracing enables ITSs to directly track and adapt instruction to student's misconceptions and unmastered knowledge as it tracks their progress through active step-by-step practice. When AI2T learns ITS behavior it induces rule-based knowledge structures sufficient for executing this historically difficult-to-build class of ITS behaviors. Model-tracing ITSs have been estimated to require as many as 200-300 developer hours per hour of instruction [2]. In the two domains that have participants author this work, AI2T cuts the most difficult programming elements of authoring down to about 20-30 minutes of effort. Authoring with AI2T is similar to tutoring a human, and involves mostly just solving problems and grading it as it solves problems.

AI2T can induce correct and complete model-tracing behavior in the sense that it induces rules that permit only the correct next actions in step-by-step problem solving, and no incorrect next actions. This induced behavior also permits the generation of "bottom-out" hints: correct next actions for problem steps that are requested by students as a last resort when they are stuck [22]. Typically other features of ITSs (which we do not focus on in this work) like requestable conceptual hints, automatic feedback messages, and knowledge-tracing (i.e. tracking students' mastery of particular knowledge) [17] are built on top of the kinds of rule-based tutor models that AI2T learns.

Readers may fairly wonder why we have not taken an LLM-based approach. Large Language Models (LLMs) have many exciting applications in education but have yet to demonstrate behaviors similar to model-tracing tutors. Insofar as out-of-the-box LLM chatbots 'tutor', they typically default to providing full step-by-step explanations similar to textbook worked examples [53]—an impressive feature no doubt, but one that is prone to abuse [14], and inconsistent with historically successful ITS designs that focus on learning-by-doing exercises. Even if an LLM is prompted or fine-tuned to

evaluate intermediate steps of a student solution in-progress, there is no guarantee that it will provide consistently accurate evaluations and, indeed, its underlying statistical inference approach makes 100% accuracy highly unlikely.

While LLMs have improved rapidly on static benchmarks of academic tasks, especially math benchmarks like GSM8K [16] and MATH [27], more incisive investigations support the notion that LLMs rely heavily on memorizing problem-solution pairs. Mirzadeh et. al. show that small perturbations in GSM8K problems cause precipitous declines in LLM accuracy [48], and recent LLM evaluations on tutoring system content—presumably out-of-distribution from static benchmarks—show impractical rates of error of at least 10% on step-by-step solution generation and grading tasks [23] In an educational setting, even a small rate of hallucinated incorrect instruction may do more harm than good. Plausible but incorrect responses are a likely recipe for producing student misconceptions. Conventional hard-coded tutoring systems are by contrast decidedly precise in execution—when properly debugged they don't make mistakes—and moreover in the best cases, they are precisely designed. Their interfaces and instructional strategies are carefully crafted around insights yielded from investigations of domain experts and students. AI2T innovates toward authoring these trustable and intentionally designed tutoring programs without writing or verifying code.

1.2 Traditional ITS Authoring

Several tools like CTAT example-tracing [2], OATutor [55], and others [5, 26, 54] offer approaches that are faster than programming-based authoring and accessible to non-programmers. Yet these methods place considerable limits on ITS control structures. OATutor supports strictly sequential "tutoring pathways" [55], and CTAT example-tracing supports graphs of states and actions that can diverge, re-converge, and manifest unordered groups [2]. Both CTAT example-tracing and OATutor enable means of mass-producing problems within these fixed control structures via a template-filling approach, where special variable strings are replaced by problem-specific content detailed in spreadsheets. In practice, this method of mass-producing problems still requires some programming effort, for instance, by writing programs to fill in the content of each step in a spreadsheet formula language [2, 55]. By contrast, authoring with AI2T requires no programming (or program checking), but can produce behaviors that are typically only implementable by programming.

The domains in which these programmed rule-based model-tracing ITSs are particularly useful include complex procedural skills that involve context-specific decision making, and multi-step domains that should permit some solution flexibility that is impractical to capture in fixed sequential or graph-based control structures. STEM-based procedural tasks are a major category of domains where these features are desirable, yet any domain that teaches complex procedural skills is a good candidate for the kinds of model-tracing behaviors that AI2T can author.

1.3 Issues with prior Authoring-by-Tutoring Approaches

Prior authoring-by-tutoring approaches like those prototyped with SimStudent [45] and the Apprentice Learner (AL) [43, 63], have demonstrated efficiency benefits over conventional authoring tools. However, in studies of these prior approaches, untrained participants [63], and in some cases also the creators of those approaches [43, 45, 64], failed to produce ITS behavior that was 100% model-tracing complete. *Model-tracing completeness* is a measure of how nearly a program imitates the behavior of a model-tracing ITS. It is defined as the proportion of reachable problem states across a large holdout set of problems where the program permits every correct next action and no incorrect actions. Weitekamp et. al reported that their authoring-by-tutoring approach with the Apprentice Leaner (AL) fell short of 100% model-tracing completeness both because of limitations in its interaction design and the learning mechanisms of their agent [63]. For instance, they report interaction design issues related to participants locating and fixing mistakes, navigating and providing feedback over diverging solution paths, and estimating when training is complete. AI2T builds on the interaction designs and machine-learning approaches of prior work to resolve many of these issues.

2 AI2T: An Overview

AI2T's central interaction loop is similar to other authoring-by-tutoring approaches. The author must:

(1) Build an HTML interface
(2) Demonstrate how to solve an initial problem
(3) Tutor the agent by grading it step-by-step over several problems and provide demonstrations when it is stumped

2.1 Interface Building

Since AI2T works on top of arbitrary HTML interfaces, the initial interface authoring process is not a focus of this work. There is no shortage of good HTML editors, including ones specifically designed for ITSs [2], and which include innovative new features like natural language to HTML translation [9]. The second and third steps are more central to this work, they are the means by which the user teaches AI2T.

2.2 Demonstration Interpretation

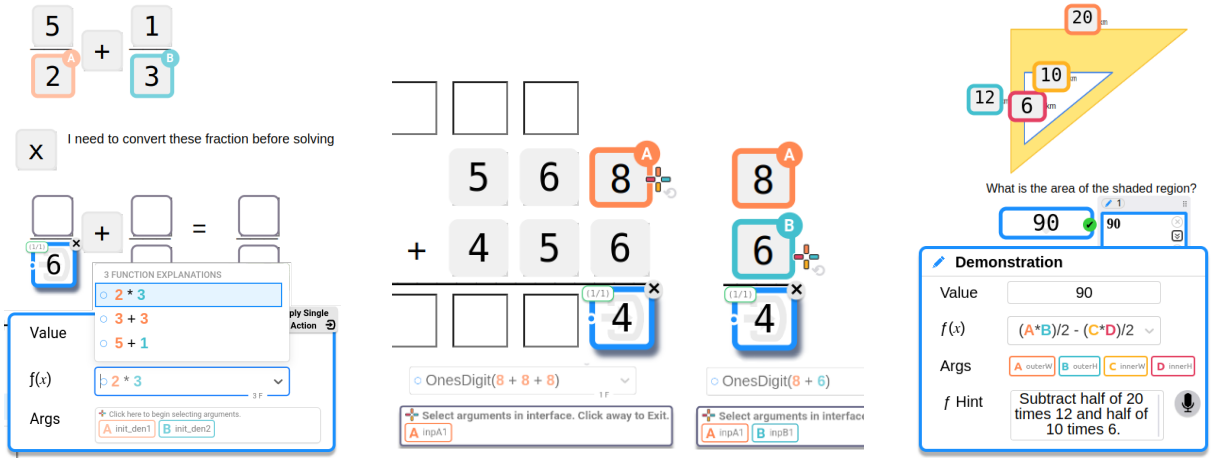
The *then-* component of the *if-then* rules of a tutoring system dictate how individual steps are performed. In authoring-by-tutoring these are learned by demonstrating a step solution. The agent then synthesizes one or more short programs that reproduce each demonstrated value in a problem solution. In AI2T the user selects among these possible programs.

2.3 Tutoring for Control Flow Induction

The *if-* component of the *if-then* rules of a tutoring system dictate its control flow: the permittable solution paths. In authoring-by-tutoring, this is learned from positive and negative examples produced by the author grading the agent's step-by-step actions.

3 Research Questions

Prior work has identified several challenges with authoring-by-tutoring [63], especially with its third phase of learning rule preconditions from interactive tutoring. AI2T solves many of these problems in part or in whole through improvements in its underlying AI and interaction design. Our research questions are as follows:



(a) (b) (c)

Figure 2: User interactions for demonstration interpretation. (a) In a fraction arithmetic tutoring system, an author has demonstrated the converted fraction 6. The agent displays several possible explanations for this demonstration in a drop-down. The arguments of the correct explanation are currently highlighted in the tutor interface. (b) In a multi-colum arithmetic tutoring system, an author narrows down the explanations of a demonstrated 4 by selecting the values they used as arguments. (left-frame) They first select the 8 and (right-frame) they select the 6. At each selection, the agent's explanation for how the 4 was generated is immediately updated. (c) In a geometry tutoring system, an author describes the formula that produced their demonstration 90 in natural language. The agent uses this description to guide the search for a formula that explains it.

- **RQ1**: To what extent are AI2T's machine learning algorithms more robust and data-efficient than prior approaches?
- **RQ2**: Does AI2T resolve previous interaction design issues with authoring-by-tutoring, such as locating and fixing mistakes and authoring multiple solution paths?
- **RQ3**: Can AI2T help users predict when they have taught it 100% complete tutoring behavior?

RQ3 has implications for interactive task learning beyond authoring-by-tutoring. When training data is limited, machine learning systems are typically poor at predicting their performance on holdout data. By contrast, AI2T is self-aware of its learning; it can make precise estimates of its performance in unseen situations, and how learning experiences affect its holdout performance (so long as those examples are not too far out of distribution). By surfacing a simple certainty score back to users, they can estimate how close AI2T is to being finished with learning.

4 Demonstration Interpretation

AI2T innovates beyond SimStudent, and AL on the second step of authoring-by-tutoring: demonstration interpretation. With sub-second latency, AI2T can search for formulae that transform values in the interface into the value demonstrated by the user by testing millions of compositions of functions chained from a corpus of primitive functions. At first, this process is unconstrained and may return a set of candidate compositions (Figure 2.a) which is sometimes too large for the user to select among. However, the user can guide AI2T's search and narrow down the set of candidate explanations by additionally selecting the correct input arguments for the target formula (Figure 2.b), or by describing the target formula mathematically or in natural language (Figure 2.c). When a formula is selected by the user it acts as the *then* component of the *if-then* structure in AI2T's learned rule-like skills.

Similar problems in action model learning [4], program synthesis, and inductive logic programming [50, 58], claim to innovate in this space by the introduction of special meta-primitives and constraints that impose biases on a wide and blind search. AI2T instead constrains search by putting a human in the loop—a human who may be ignorant of the contents of AI2T's function library (which includes hard-to-anticipate functions like OnesDigit(x) or GreatestCommonDenominator(x)) yet can guide the search for a solution through natural language and argument identification. With these tools, the user can zero in on an appropriate formula with natural tutoring interactions instead of needing to write formulae from scratch.

This method has been reported in prior work [redacted], so it is not our emphasis here. Yet for the sake of completeness, we note that these interactions have been well-received by the pilot testers and the study participants of this work.

We leave an important subproblem in this space to future research: what is a user to do if they need a primitive that AI2T does not already have? This is of course an issue faced by all software shipped with a preset function library; from ITS authoring tools to spreadsheet editors. LLM code generation could play a role in a solution while retaining AI2T's emphasis on non-programmer support. LLMs have certainly been helpful for expanding AI2T's function library to its current state. We leave this subproblem to future work.

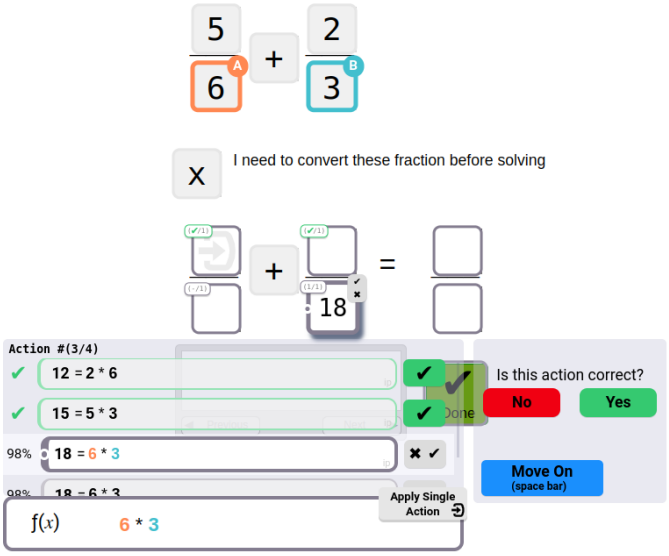


Figure 3: The action window shows each of the actions proposed by the agent. Currently, action 3 of 4 is selected. The proposed actions convert the expression 5/6 + 2/3 to have a common denominator 18.

5 Teaching Control Flow with Interactive Tutoring

Tutoring system authoring suffers from a challenging design problem: the author must attend to the branching of alternative permittable solution paths, yet normal problem-solving proceeds by sequential steps. Prior authoring-by-tutoring approaches [63] have struggled with usage patterns where authors validate just the first correct action suggested by the agent (similar to grading a human student), but neglect to grade actions to teach the agent alternative solution paths that should be permitted by its tutor-model.

AI2T supports authors in visualizing and verifying alternative proposed solution paths: 1) by presenting predictions of multiple correct next actions in each intermediate tutor state, and 2) by enabling users to visualize and navigate between alternative solution paths.

5.1 Supporting Complete Feedback in each Tutor State

The action window (Fig. 3) frames AI2T's central interaction loop, whereby the author grades each of AI2T's several proposed next actions. Authors can visualize each action by selecting or hovering over each item in the action window. Clicking the ✗ or ✓ icons on the toggler (Fig. 3) in the action window assigns negative or positive feedback to each action. The user is prompted to provide feedback to any ungraded actions: "Is this action correct?". When all actions are graded it asks: "Demonstrate any other actions for this step. Press **Move On** to apply these actions". This reminds the author to demonstrate any correct next actions that AI2T has not already proposed.

Small indicators overlaying the tutor interface give a bird's eye view of which interface elements AI2T has proposed acting upon in each tutor state and offer an alternative place to click or hover to preview or select actions.

Color and icon changes signify the user's progress toward grading each of AI2T's proposed actions. The author must change the ungraded actions (grey), into positively graded (green ✓) or negatively graded (red ✗) actions. Any user-demonstrated actions appear in blue. This pallet uses a particularly light green, dark red, and slightly blue-tinged grey, to reduce mix-ups common to various forms of color-blindness. We verified this pallet using an application that simulates protanopia, deuteranopia, and tritanopia and double-checked with colorblind labmates. Selected actions additionally use color to highlight their arguments (e.g. 6 * 3, in Figure 3).

5.2 Supporting Solution Path Navigation

AI2T automatically generates graphs that help authors visualize AI2T's tutoring behavior, and track the solution paths that they have already trained it on for each problem. These behavior graphs enable authors to navigate between different problem states. Since an AI2T agent learns hierarchical rule-like knowledge structures (not graphs), each generated behavior graph is simply a visualization of the agent's induced program applied to a particular problem—not a direct visualization of its internal knowledge structure. The behavior graph shows all of the diverging action sequences that the agent believes are correct. This allows the author to navigate between problem states by clicking on particular nodes, and select particular actions from the edges.

Users can pan through AI2T's generated behavior graph by scrolling or dragging, and zoom in and out with shift+scroll. Entering a new state for any reason including clicking on a node, selecting an edge in a different state, or applying an action in the main interface, automatically animates the graph view so that it is

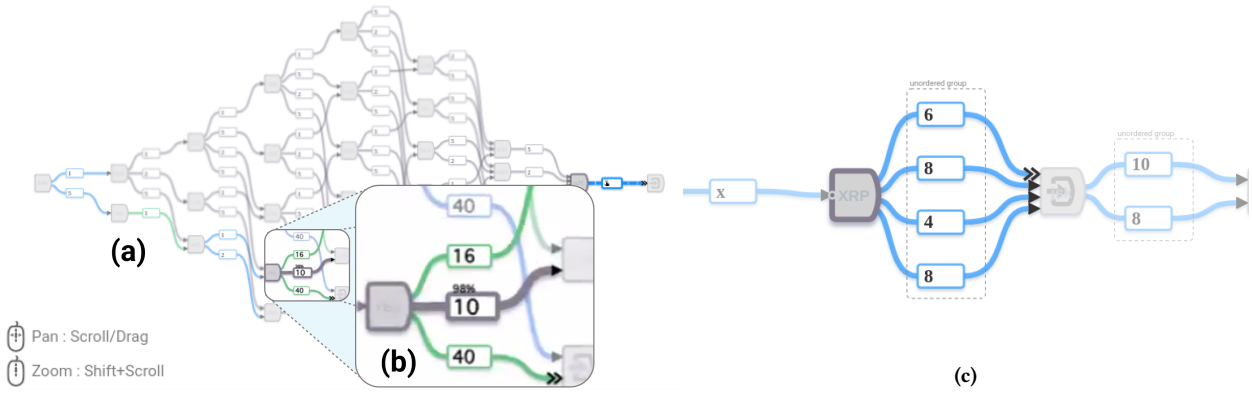


Figure 4: (a) A generated behavior graph and (b) a behavior graph zoomed into the current state. Of 3 proposed actions, the 1st and 3rd have been given positive feedback and the 2nd is selected. Demonstrations are blue, actions with positive feedback are green, and grey edges are proposed actions that still need feedback. (c) A revised implementation of the behavior graph visualization that displays unordered groups in place of permutated action sequence like in (a).

centered on the new state and its downstream actions. This feature keeps the graph aligned with the current problem state.

Our user testers did not have much trouble with navigation with these graphs. However, many users found it disorienting when the same actions occurred more than once in the graph but in different orders because of step-order flexibility. To resolve this issue we introduced a feature where AI2T explicitly learns unordered groups as part of hierarchical task network induction (described later in section 4.1). These groups are displayed within the behavior graphs, considerably simplifying their structure so that one edge in the graph tends to correspond to just one unique action. This feature also sped up authoring by reducing the number of problem states that authors needed to grade AI2T in.

5.3 Visualizing Action Certainty

Finally, within the action window, and on each edge of the behavior graph visualization, AI2t reports a continuous certainty score ranging between -100% and 100%. These scores indicate how certain the AI2T is that each proposed action is correct or incorrect. Negative values indicate that the agent is mostly certain that an action is incorrect and positive values indicate varying degrees of certainty that the proposed action is correct. As we describe in section 8.2, these values come from STAND's *instance certainty* measure which we will show is a fairly reliable indicator of prediction certainty and a good indicator of AI2T's actual learning progress—something we show is not true of many alternative methods of estimating prediction probability. Roughly speaking, if the agent proposes only actions with 100% certainty scores for a particular problem state, then this is a fairly strong indication to the author that the agent will exhibit 100% model-tracing complete behavior in similar situations. Mixtures of lower certainty scores are a fairly strong indication that the user should continue to train the agent on more problems.

Figure 5: Three actions proposed in the action window with certainty scores (49%, 49%, and 66%)

6 AI2T's Generality

Prior simulated learner systems have illustrated that they succeed (or usually just partially succeed) at learning to solve or replicate tutoring behavior in a variety of domains. Prior work with Sierra [61], SimStudent [45], and the Apprentice Learner [42], are similar to but less robust than AI2T, have trained agents on dozens of domains including algebra, stoichiometry [39], geometry [10], simple linguistic tasks like Chinese character translation and article selection [41], a few games, and other arithmetic tasks like subtraction [61] and multiplication.

Our internal evaluations of AI2T have shown that it can replicate (with 100% model-tracing completeness) several existing tutoring systems using its current primitive library, including 10 from the Cognitive Tutor MathTutor suite (middle school math) [1], and another 30 from Apprentice Tutors (algebra and pre-calculus) [24]. Between these two paradigms, the Cognitive Tutor MathTutors exhibit the interesting varieties of control-flow-based solution flexibility that we emphasize in this work. They take about 30 minutes to author with AI2T. The relatively more advanced topics from the Apprentice Tutors, are comparatively simple in terms of permitted solutions; demanding particular sequential steps in most cases. It takes just one or two demonstrated solution sequences to replicate these domains with AI2T.

Often solution flexibility in an ITS is more a function of an author's design mentality than it is a function of the basic requirements of solving problems from start to end. A fundamental design perspective in the early days of ITSs was that the tutoring system was a means of developing student's knowledge to align with an expert system: a collection of meticulously hand-written rules, that replicate the behaviors of experts, surfaced through methods of knowledge elicitation like cognitive task analyses [15]. A regular outcome of cognitive task analyses is that the scope and complexity of an expert's tacit knowledge usually far exceeds what they can describe by direct description [68]. Typically, much of the forgotten details pertain to special considerations employed in edge cases, and the criteria for deciding between different courses of action (i.e. the *if* component of *if-then* rules). Our special focus here on AI2T's ability to robustly learning skills that require contextual decision-making echoes this history. We solve here a subproblem that is foundational to the development of tutoring systems that support the mastery of true expertise instead of just the practice of rote sequential steps.

7 User Study Domains

As our focus here is on algorithmic advances (**RQ1**), basic usability (**RQ2**), and testing the feasibility of novel interaction techniques (**RQ3**), we limit our selection of target domains to ones that balance representation in prior literature, complexity, and wide familiarity. The third is essential as it allows us to attribute user errors to interaction issues instead of to mathematical mistakes.

Our simulation experiments and user studies test users as they author with AI2T on two domains: multicolumn addition and fraction arithmetic. Both domains require some contextual decision-making and permit step-order-based solution flexibility. Expressed as heirarchical procedures both domains involve deciding between alternative subprocedures based on the context of the problem. Multicolumn addition has been tested in prior authoring-by-tutor approaches [63] and is a good example of a domain where an author must almost always resort to programming the ITS behavior in place of conventional graphical authoring tools since the solution steps vary considerably between different problem instances.

In multicolumn addition, students practice the algorithm for summing large numbers together by computing partial sums and carrying their tens digit (if necessary). The ground-truth model-tracing behavior permits add and carry actions to be applied in either order. The main difficulty of this domain in terms of inducing the correct behavior is that a contextual decision must be made about when to carry a 1 or not, and for always adding three numbers instead of two if a 1 was carried from the previous column. To replicate prior work [63] we limit this domain to problem instances that have pairs of 3-digit numbers.

In fraction arithmetic, students apply one of three arithmetic procedures (add, multiply, or convert-then-add). This design was conceived in response to data that showed that a large proportion of students' fraction arithmetic mistakes pertain to applying incorrect fraction arithmetic procedures, and not necessarily to applying each procedure individually [56]. The tutor partially scaffolds this decision by first asking the student if they "need to convert these fractions before solving". The tutor is flexible in terms of the step order of filling in the digits of the converted fraction, and final combined fraction. In both tutors, the student must press a 'done' button as a final action.

8 AI2T: Mechanisms for Self-Aware, Data-Efficient, and Robust Induction

Prior authoring-by-tutoring approaches have used simulated learners that simulate human-like induction [42, 45, 61, 67] from demonstrations and supervised correctness feedback. These simulated learners largely share a similar breakdown of 3 core learning mechanisms that enable data-efficient induction that can be taught interactively. These mechanisms collectively induce several production-rule-like skills, that execute step-by-step solution strategies and flexibly track students' solutions in an ITS. Unlike hand-programmed production rules, skills are refined over the course of interactive training to reflect the author's instructed behaviors.

Each of the typical 3 core mechanisms in simulated learners used for authoring-by-tutoring induces different kinds of generalizations within each skill: 1) compositions of primitive functions that express *how* skills produce actions from other information in an interface, 2) patterns or concepts that capture *where* each skill might locate candidate inputs and outputs, and 3) preconditions that express *when* as in what contexts a potential candidate application of a skill is a correct application of the skill. The mechanisms that learn these different types of generalizations are typically referred to as *how-*, *where-*, and *when-learning* mechanisms respectively [45, 63].

AI2T's learning-mechanism implementations are similar to prior implementations of AL [63] with the exception of two important innovations. AI2T uses an algorithm called STAND for *when-learning* (i.e., precondition induction), and introduces a fourth learning mechanism which we call *process-learning* that organizes induced skills into hierarchical task networks (HTNs). Unlike methods that have users describe HTN structures directly in a top-down manner [29, 36, 40] AI2T learns HTNs directly from authors' demonstrated action sequences (i.e. via bottom-up induction). *Process-learning* requires no new interactions from authors. Authors still only need to solve problems and grade AI2T as it solves problems. The author does not need to plan or verify any part of *processs-learning*'s induced HTNs. In our user studies (section 6) we do not even display AI2T's induced HTNs to participants.

8.1 *Process-Learning*: Hierarchical Task Network Induction from Action Sequences

AI2T's *process-learning* mechanism induces HTNs that recursively break tasks into subtasks that terminate in primitive skills that take individual actions. Methods in the HTN are higher-order skills that carry out tasks as ordered or unordered sequences of sub-tasks and primitive skills. Figure 6 shows an HTN that AI2T induced for the fraction arithmetic ITS. The task "Combine Fraction Expression" can be achieved by three different disjoint methods: method 2 adds like fractions, method 3 converts fractions to have the same denominator before adding them, and method 4 multiplies fractions. The induced preconditions for each of these methods from when-learning (not shown in the figure) gate their consideration as acceptable solution strategies. In this example, the correct preconditions for methods 2, 3, and 4 would make them mutually exclusive.

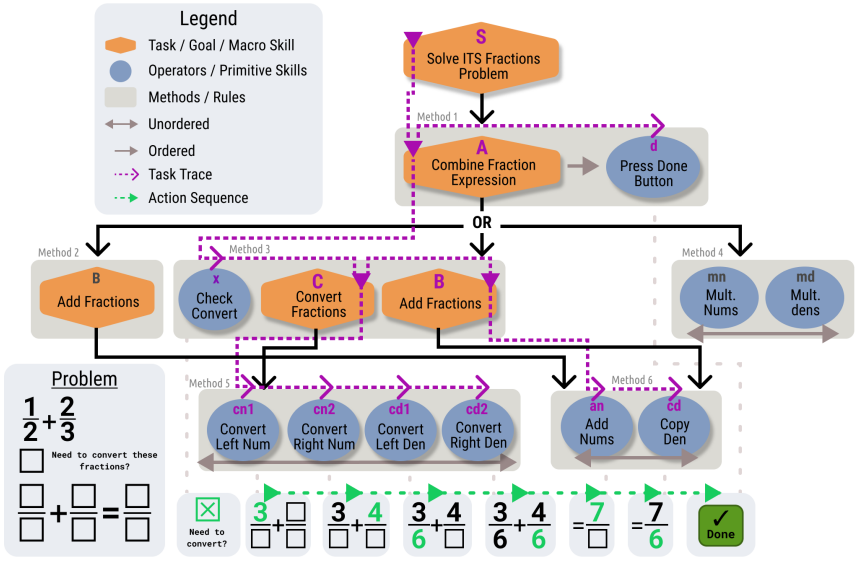


Figure 6: A Hierarchical Task Network (HTN) learned by AI2T for the fraction arithmetic tutoring system. A task trace (dotted-purple) and action trace (dashed-green) of solving an addition problem with fractions with different denominators.

However, in problems with multiple valid approaches to achieving the same subtask, multiple methods may be applicable simultaneously, representing an alternative solution strategy permitted by the tutoring system. For instance, Figure 6 captures just one method (#5) for the "Convert Fractions" subtask, but an author could certainly add another to the HTN by demonstrating it at the appropriate problem step.

Learning correct ordering constraints with the typical 3-mechanism approach without *process-learning* (like SimStudent and AL) requires the author to give agents negative feedback when they suggest applying individual skills out-of-order; often across dozens of problems. For instance, prior authoring-by-tutoring agents built with AL [63] have struggled to strictly sum multi-digit numbers from right to left until authors first corrected several out-of-order mistakes. Inducing skill preconditions with the typical 3-mechanism simulated learner approach often requires collecting diverse sets of negative examples.

Adding *process-learning* (for a 4-mechanism approach) considerably simplifies the learning of ordering constraints. *Process-learning* situates primitive skills into methods within a hierarchical process. When AI2T first sees an author solve a problem with a particular action sequence, it assumes that all solutions should proceed in strictly that order. When AI2T observes additional solutions that deviate in their action order, or use entirely new approaches it may relax its ordering constraints or add new methods to its HTN. As authors demonstrate new forms of solution flexibility AI2T's HTN introduces new subtasks and methods to accommodate them.

An important feature of AI2T's *process-learning* mechanism is that its HTN induction is agnostic to lesson order. Prior simulated learner approaches that induce HTNs, like VanLehn's Sierra system [60, 61], have required curated lesson sequences or suffer complete inductive failure. By contrast, AI2T will induce the same HTN structure regardless of the sequence in which new problems are presented, meaning authors do not need to explicitly design good lesson plans for AI2T.

AI2T's HTN representation language also allows for the induction of HTNs with optional or conditional symbols in methods. For instance, carrying a 1 in a multi-column addition problem is a conditional step—it only occurs for partial sums greater than 10. An example of an optional step may be writing the numbers that are added or subtracted from each side of an algebra equation before calculating the next line—an author may not want to penalize students who explicitly skip this step, but still track student solutions and provide adaptive supports in cases where that step is not skipped. Similarly, recent work has promoted HTN-based tutor models as a favorable ITS design paradigm [59]. HTNs naturally capture knowledge at multiple levels of granularity, and thus are a good knowledge representation for ITSs that support adaptive scaffolding where incremental step-by-step tutoring support is faded as students become more proficient [31, 47, 57]. We do not evaluate this particular feature in this work, but AI2T's HTN induction approach is certainly conducive to supporting it.

8.2 STAND: Self-Aware Precondition Induction

In prior work, *when-learning* has been identified as a major limiting factor for efficient authoring [63]. Typically *how-* and *where-learning* converge to their final generalizations from just one or two examples. *When-learning* requires several additional correct and incorrect examples to induce correct pre-conditions for each skill. Our introduction of *process-learning* in AI2T, greatly simplifies the burden of *when-learning* by situating each primitive skill in methods that can have strictly ordered or unordered constituents. In prior work, *when-learning*'s induced pre-conditions needed to control the order in which primitive skills were applied. With *process-learning* order constraints are mostly handled by the

HTN, and *when-learning* induces pre-conditions that simply select between alternative methods in the HTN.

Independant of *process-learning*, STAND is more data-efficient at precondition induction than prior when-learning approaches [66]. Prior approaches include fitting decision trees [43, 63] or applying inductive logic programming approaches [45] to learn each skill's preconditions. Approaches like these learn a single best set of preconditions that select correct actions and reject incorrect ones. By contrast, STAND learns a space of preconditions that are consistent with the authors' positive and negative training examples. This space accounts for a complete set of *good* candidate preconditions instead of arbitrarily selecting single sets of preconditions by breaking ties within a greedy construction process randomly. In this way, STAND explicitly models the inherent ambiguity of trying to learn preconditions that perform well on unseen examples—something that is especially challenging in an interactive training setting with limited training data.

By inducing a space of generalizations, STAND is conceptually similar to a version space. However, STAND does not suffer from "version-space collapse", where induction completely fails from noisy data, nor does it restrict conditions only to conjunctive expressions [49]. STAND can learn arbitrary disjunctive normal logical statements and when combined with AI2T these statements can be expressed with relational (i.e. variablized) predicates. AI2T enables relational condition learning by restating problem state features before they are passed to STAND. Features are expressed relative to the interface element being acted on, and the interface elements that were selected as arguments by each skill's matching pattern (induced by *where-learning*). Importantly STAND is very computationally efficient, meaning it does not add precievable latency time between authors' interactions.

In addition, STAND can produce a measure (from -100% to 100%) called *instance certainty* [66] that estimates how certain it is of predicting the correctness label of an unseen example. STAND's instance certainty on single unseen examples tends to increase when its holdout set performance increases, meaning it is a good heuristic for estimating actual learning gains.

Conventional methods that can estimate their prediction probability including ensemble approaches like random forests and XGBoost lack this property; as we demonstrate in the following section. STAND's *instance certainty* measure is more effective than these alternatives because it captures how unambiguous the label prediction of an example is given all of the sets of preconditions that capture the example. Low *instance certainty* indicates high disagreement among STAND's space of possible preconditions. High *instance certainty* reflects high agreement. Ensemble methods operate similarly, but not nearly as comprehensively. The countable constituents of an ensemble don't characterize all *good* candidates, just several stochastically constructed ones.

9 Simulation Experiments

In addition to testing AI2T with users, we wanted to ensure that STAND and *process-learning* produce improvements in learning efficiency over prior approaches. In these experiments, we first evaluate STAND using it for *when-learning* in a typical 3-mechanism simulated learner configuration (with *how-*, *where-*, and *when-learning*, but not *process-learning*). We compare STAND to various alternative *when-learning* approaches using an automated training system that mimics the demonstrations and feedback of an ideal author. We apply this training approach in the two domains we had participants author in our user study (section 10): multicolumn addition and fraction arithmetic.

In this setup, each agent receives ideal on-demand demonstrations and correctness feedback. In each state all proposed next actions are labeled as correct or incorrect. If an action is missing then it is demonstrated to the agent with annotations that make the underlying reason for the action unambiguous. Each demo is annotated with the formula for producing the action's value, and the arguments used. This replicates the behavior of an ideal user who always selects the correct formula to explain each demo from among the suggested possibilities. These annotations enable *how-* and *where-learning* to produce errorless generalizations almost immediately, meaning essentially all errors can be attributed to *when-learning*. No annotations are provided to assist *when-learning* besides the correctness labels of each action. Just like an ideal author, the training system trains the agent on all alternative solution paths for each problem.

We compare several classifiers with STAND:

(1) **Decision Tree**: A decision tree using gini impurity [8] as the impurity criterion.
(2) **Random Forest**: Scikit-learn's implementation of random forest ensembles [7]. Random forests use bagging [6] to independently train several decision trees on subsets of the data. We configure the random forests to use 100 trees.
(3) **XG Boost**: An ensemble method that trains multiple decision trees one at a time. This method uses gradient-based sampling to re-weight the samples for subsequent trees [12].

These tree-based methods are chosen because they excel at learning from small datasets of structured data. In all models no limits are set on tree depth or leaf size since for these condition-learning tasks the available features from the tutoring interface are sufficient for separating correct and incorrect candidate actions perfectly. Since the features selected by ideal preconditions should be noiseless, the trees will already tend to be no more complex than the ideal solution, and limiting their depth could only prevent that solution from being discovered. The two ensemble methods are included for comparison with STAND, and to compare the utility of their prediction probabilities with STAND's instance certainty estimates. Each model is re-trained on 40 repetitions over a sequence of 100 randomly generated problems.

9.1 Productive Monotonicity: Certainty Score Change vs Holdout Set Performance Change

Productive monotonicity is defined as the proportion of changes in certainty estimates over holdout set actions that increase (toward 100%) when the action is correct and decrease (toward -100%) when the action is incorrect. High productive monotonicity reflects the degree to which changes in certainty estimates mirror actual learning gains (i.e. increases in holdout set performance).

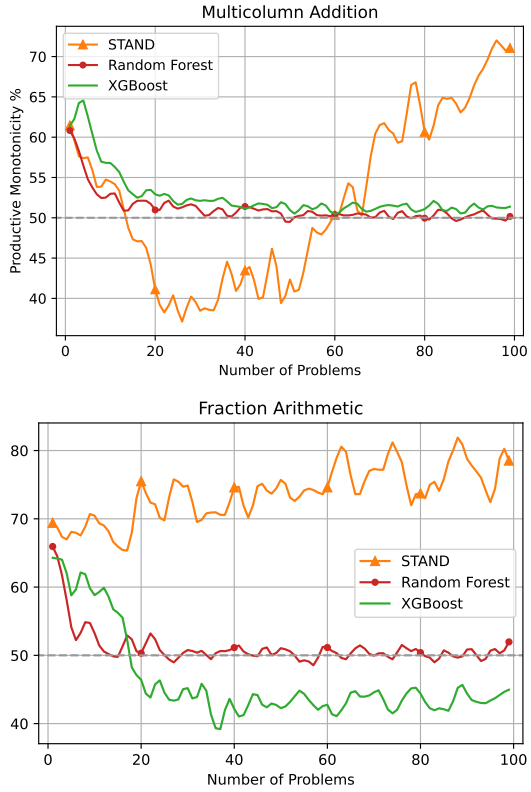


Figure 7: Productive Monotonicity By Problem

Our results show that STAND's instance certainty measure has considerably higher overall productive monotonicity than the two ensemble methods' prediction probabilities. Thus, increases in instance certainty reflect actual increases in holdout set performance, meaning instance certainty is a relatively good heuristic for determining when STAND has learned. By comparison, the random forest and XGBoost's prediction probabilities align with actual changes in holdout performance only about 50% of the time—they are not much better than chance.

In multi-column addition, STAND's productive monotonicity is $< 50\%$ for the first 60 training problems and $> 50\%$ thereafter. This pattern may occur in this domain because in the early stages of training the space of generalizations that STAND encloses is still growing (and entertaining new disjunctions) as new edge cases are encountered. Fractions may not show a similar pattern because purely conjunctive preconditions tend to suffice in this domain, and so STAND's space of generalizations tends to strictly shrink monotonically throughout training.

9.2 Precision at High Certainties

If a *when-learning* classifier predicts that an action is correct with a high certainty of 90%-100% then there should be a very low probability that the action is incorrect.

Table 1: Total Precision at High Certainties

	MC Addition		Fractions	
	$\geq 90\%$	$= 100\%$	$\geq 90\%$	$= 100\%$
STAND	93.19%	99.81%	95.70%	100.00%
Random Forest	97.15%	94.79%	95.19%	93.72%
XGBoost	98.35%	100%	99.39%	100.0%

Our simulations show that XGBoost has the highest precision at high certainties. For predictions of 100% STAND is nearly as precise as XGBoost in multicolumn addition and equally 100% precise in fractions. For predictions of $\geq 90\%$ STAND's precision is closer to 90%, which is arguably a desirable property—as it indicates some alignment of instance certainty with actual ground-truth precision. These results validate that certainty scores of 100% are typically only given to actions that are truly correct, and that certainty scores between 90% and 100% tend to apply to actions with a small probability of error.

9.3 Per-Problem Completeness

Finally, we verify that together, STAND and process-learning produce more data-efficient learning and higher rates of 100% model-tracing completeness on holdout data. We report each model's model-tracing completeness on a holdout set of 100 problems, evaluated at the end of each training problem.

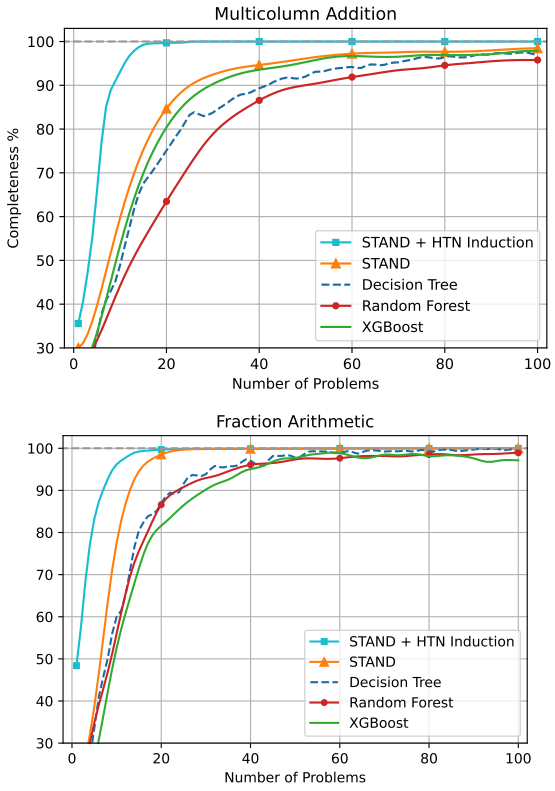


Figure 8: Average holdout completeness by problem.

Table 2: Average Holdout Completeness at Problem N, and Number of 100% Complete Repetitions at problem 100.

	Multicolumn Addition				Fractions			
	N=20	N=50	N=100	100% Reps	N=20	N=50	N=100	100% Reps
STAND + HTN Induction	**99.72%**	**100.0%**	**100.0%**	**40/40**	**99.69%**	**99.94%**	99.99%	38/40
STAND	85.45%	96.10%	98.62%	19/40	98.72%	99.91%	99.99%	38/40
Decision Tree	75.75%	91.86%	96.97%	10/40	88.15%	97.24%	99.88%	38/40
Random Forest	64.16%	90.02%	95.53%	0/40	88.13%	97.44%	98.97%	11/40
XGBoost	81.20%	95.40%	98.01%	3/40	81.12%	96.20%	97.34%	27/40

In both domains, STAND's average completeness is higher than the competing models throughout the training sequence. This implies that STAND has better data efficiency and asymptotic performance since it can achieve greater levels of completeness with fewer training problems. In 19 of 40 MC addition repetitions STAND achieved 100% completeness after training on a sequence of 100 problems compared to 10 of 40 repetitions for decision trees. In fractions, 38 of 40 repetitions achieved 100% completeness with STAND and decision trees. The relative performance of the decision tree, random forest, and XGBoost varies between domains. Notably the random forest was the worst in multicolumn addition, likely because its bagging approach of sampling subsets of the data had the effect of dropping important edge cases, which are particularly important in this domain.

Introducing *process-learning* so that AI2T induces HTNs instead of independent primitive skills improves AI2T's learning even further. In this configuration, AI2T's absolute holdout set performance is strictly higher than the competing models throughout training, and in every single training repetition 100% model-tracing completeness is achieved in multicolumn addition. These results validate the absolute performance benefits of STAND and show empirically how process-learning promotes model-tracing completeness beyond what can be achieved with the typical 3-mechanism approach used by prior authoring-by-tutoring systems.

10 User Studies

We evaluated AI2T in two studies where 10 participants per study authored multicolumn addition and fraction arithmetic. Participants were tasked with tutoring AI2T until they believed it would exhibit correct tutoring behavior for any problem in that domain. When participants self-reported that they tutored AI2T on enough problems we scored their agents' model-tracing completeness on a large holdout set of 100 problems. Participants moved on to the next domain after seeing their scores. Participants were not permitted to tutor AI2T more to get a better score.

The central aim of these studies was to evaluate what configurations of the agent and interface design best support authors in teaching AI2T to induce correct and complete programs. Beyond qualitative observations of usability, a core element of this evaluation is to determine whether our interaction design supports authors in building model-tracing complete tutoring systems.

10.1 Methods

10.1.1 Recruitment. Each of our 20 participants was recruited via email or Slack from within our professional network. Users engaged in these studies remotely via Zoom and consented to be recorded as they screen-shared while working with AI2T. For participation in these IRB-approved studies users were compensated with a $30 Amazon gift card. We limited all sessions to a maximum of 90 minutes. No participants had used AI2T prior to these investigations.

10.1.2 User Instruction. In both studies, participants authored multicolumn addition and then fraction arithmetic. Participants received a short tutorial on how to use AI2T: we showed them how to demonstrate each step of the problem 777+777, and showed them how to give feedback to the agent on a subsequent problem 222+222. Prior to having participants begin authoring each domain, we described the behavior that we expected the final tutoring system to have, and asked that participants engage in a think-aloud: "Say whatever you are thinking as you work with the tool". Participants always began authoring with a blank agent with no prior training. Participants were provided with a blank interface for each domain, they were not required to make their own.

While users worked with AI2T we made ourselves available to answer questions. We refused to give participants any advice concerning when they should stop training the agent. We limited ourselves to suggesting that they train the agent on a variety of problems and suggested that they should at least keep training AI2T until it seemed like it had stopped making mistakes.

At the end of each session, we asked users to give their open-ended feedback about their experience. We simply asked: "What worked well and what didn't work well as you were using the tool?". In study 2 we also explicitly asked participants if they noticed the certainty score indicators and whether they considered them when deciding whether or not to stop training the agent. To validate that our AI2T is usable by non-programmers we had study 2 participants report on their level of programming experience on a Likert scale from 1-5.

10.2 Study 1: Piloting an In-Development Version of AI2T

Study 1 was conducted to assess the efficacy of an in-development version of AI2T. This version of AI2T was configured to replicate the 3-mechanism agent configuration used by SimStudent and AL. This early variant of AI2T utilized STAND but lacked a *process-learning* mechanism. Study 1 participants were not particularly successful at authoring robust tutoring systems. However, observing their difficulties yielded insights that informed the final design of AI2T for study 2.

Three things are missing in study 1 version of AI2T 1) the agent has no *process-learning* mechanism, 2) the behavior graph does not display unordered groups so the graph edges diverge combinatorially, and 3) there is no action window so authors must use the

edges of the generated behavior graphs or the indicators in the tutor interface to locate and switch between different proposed actions. In this version, certainty scores only appear over edges in the behavior graph, since there is no action window.

Table 3: Results for Study 1

	MC Addition		Fraction Arithmetic	
User#	Completeness	Minutes	Completeness	Minutes
1	90%	55	-	
2	99.80%	33	64.69%	32
3	71.43%	31	100%	24
4	100%	33	92.76%	34
5	91.45%	24	100%	33
6	90.60%	24	38.05%	32
7	100%	22	85.75%	35
8	100%	23	95.29%	23
9	99.15%	28	76.43%	29
10	57.07%	50	-	-
Mean	90%	30.33	82%	30.25
Median	95%	28	89%	32

10.2.1 Quantitative Results. The quantitative results for study 1 are outlined in Table 3. For multicolumn addition, 3 of 10 participants taught agents that achieved a 100% model-tracing completeness score on the holdout set of 100 random problems. Two of our participants took more than half of the allotted 90 minutes for the first domain, and we did not have them complete the second. In fraction arithmetic just 2 of the 8 participants achieved 100%. In both domains, the total authoring time—the time between beginning authoring and self-reporting that they believed the agent had achieved correct and complete behavior—was about 30 minutes.

10.2.2 Qualitative Results. Automatic behavior graph generation is one major improvement in AI2T over prior authoring-by-tutoring interaction designs [45, 63]. Study 1 users generally had little trouble panning and selecting states and actions in the behavior graph. However, without unordered group induction, the study 1 behavior graphs displayed every permutation of action order as a distinct path in the graph. This greatly inflated the number of problem states displayed to the user and encouraged an authoring strategy whereby users graded actions along every permutation. This approach was time-consuming and not strictly necessary (grading a small subset of the paths would have sufficed). Some users were disoriented when previously graded actions reoccurred in parallel paths as ungraded actions. Navigating to these edges gave them the impression that their previous effort had been undone, as they needed to verify the same actions again.

Without an action window, the study 1 interface showed certainty scores above each behavior graph edge. This presentation did not appear to be effective as none of the users mentioned the certainty scores in their think-aloud or follow-up interviews.

Interaction issues notwithstanding, study 1 participants were fairly positive about our in-development version of AI2T. 8 of the 10 participants in study 1 had used other ITS authoring tools before, including CTAT's example-tracing authoring tool. In follow-up interviews, several participants commented that they found AI2T easier to use than CTAT example-tracing because after demonstrating solutions to a single problem, the agent would suggest step-by-step solutions automatically for the remaining problems. For instance, one participant remarked "This is a lot nicer than CTAT… I like that it mostly does the problems for you." These users commented that checking the agent's step-by-step solutions was much easier than demonstrating several problems themselves, and noted that teaching AI2T seemed to take less time and effort than filling out step-by-step problem solutions in a spreadsheet.

10.3 Study 2: Assessing the Revised AI2T

As we showed in simulation (section 9) the introduction of *process-learning* makes AI2T far more effective at achieving 100% model-tracing completeness in both of our user study domains. However, a short set of pilot tests prior to study 2, with 4 participants showed that many of the user interaction issues observed in study 1 remained despite this improvement. Compared to study 1, study 2 adds *process-learning* mechanism and two interface improvements: unordered groups in graphs, and the action window. Taken together study 1 and 2 form a loose pseudo-experiment, in which study 1 establishes a baseline with several issues and study 2 implements several fixes to remedy those issues.

10.3.1 Participant Pool. For study 2 we broadened recruitment to include participants with a wider variety of backgrounds. The participants for study 2 included 5 graduate students from Anonymous University A, 3 of which specialized in educational technology, 2 human-computer interaction graduate students, 4 biology graduate students from Anonymous University B, and 1 professional specializing in the authoring of instructional technology for a major ITS project unaffiliated with University A or B. Participants' self-reported genders were equally male and female. On a 1-5 Likert scale-based self-assessment of programming experience, 6 of 10 participants reported a 2 out of 5. We consider a 1 or 2 to be a non-programmer. These 6 non-programmer participants included our 4 biology graduate students, and the 2 HCI graduate students, both of which specialize in design.

10.3.2 Quantitative Results. Table 4 outlines the results for study 2. 8 of 10 participants succeeded at training agents that achieved 100% holdout completeness for multicolumn addition. Prior work in this domain had reported lower median model-tracing completeness rates of 92%, with no instances of 100% [63]. Our study 2 results also showed users completing training in about half the time compared to prior work: a median of 22 minutes instead of 41 minutes. Reviewing the recordings of the two participants who did not reach 100%, both made mistakes during training that they did not succeed in tracking down and fixing.

In fraction arithmetic, participants self-selected all of their own problems, and 5 out of 10 participants succeeded at training agents that achieved 100% model-tracing completeness. The two lowest-performing participants made mistakes during training that prevented them from achieving more than 50% completeness (the same two who made mistakes in multicolumn addition). The median completeness in this domain was 99%. Participants trained the agent on 9 to 21 problems in this domain in 16 to 32 minutes with a median of 22 minutes.

In our follow-up interviews we asked participants if they noticed the certainty score indicators, and whether they considered them

Table 4: Results for Study 2

		MC Addition (normal)			Fraction Arithmetic				
User#	Prog. Exp.	Completeness	Minutes	N prob.	Completeness	Minutes	N prob.	Notice	Use
1	2	100%	22	13	100%	21	20		
2	2	100%	20	9	100%	32	18	✓	✓
3	2	100%	30	11	96.31%	30	16		
4	5	100%	14	11	88.52%	17	13	✓	✓
5	2	90.96%	30	14	40.18%	19	9		
6	5	100%	22	11	98.87%	16	10		
7	5	100%	28	11	100%	27	14	✓	✓
8	3	89.16%	36	11	38.73%	18	14		
9	2	100%	22	10	100%	25	18	✓	✓
10	2	100%	18	9	100%	23	21	✓	
Mean	3	98%	24.2	11	86%	22.8	15.3	5/10	4/10
Median	2	100%	22	11	99%	22	15		

when deciding when to stop training the agent. 5 of 10 participants said that they did notice the certainty scores, and 4 indicated that they considered them when deciding whether or not to stop training the agent. Specifically, these participants indicated that they took the presence of a low certainty action as an indication that the agent needed additional training on similar problems. 3 of these 4 participants achieved 100% model-tracing completeness in both domains.

10.3.3 Qualitative Results. In our follow-up interviews several participants remarked on how quickly they learned to use our tool, and how they could succeed at using AI2T to author two tutoring systems in less than an hour. As in study 1, several users remarked on how quickly the agent was able to learn from their instruction and how the authoring process became much easier once they entered the stage of mostly checking the agent's behavior on new problems. For instance, one of our biology graduate student participants remarked: "This is wild, I could teach it all that math in like 20 minutes [per topic]."

While some study 1 participants had commented on issues with the smoothness of the "interaction loop", very few study 2 participants had constructive negative feedback. Our observations of users led us to believe that the inclusion of the action window in the study 2 design was helpful in this regard. Some study 1 participants jumped between problem states without fully giving feedback to all of the agent's proposed actions, whereas study 2 participants very consistently fell into a pattern of looking through actions in the action window and giving them all feedback before moving on. The inclusion of unordered groups meant that there were far fewer states in the study 2 version for users to navigate through. For instance, much of the users' time in study 1 was spent going through many diverging states in large behavior graphs, especially for the fraction arithmetic domain, but unordered groups spared study 2 users from the tedium of grading combinatorial paths. While some users in study 1 expressed that they had become disoriented navigating between problem states, study 2 participants did not indicate any similar issues.

Our follow-up interviews provided strong evidence that the users who noticed the certainty scores used them successfully to gauge when they should stop training the agent. For instance, participant 2 in study 2 said, "I definitely would have stopped teaching it earlier if I hadn't seen the low confidence on some problems that I thought it already knew how to do."

10.4 Discussion

Overall our study 2 results show that our redesign produced a considerable improvement over study 1. Half of the study 2 participants succeeded at training agents with 100% complete tutoring system behavior on both domains, usually in under half an hour. Our interviews with users also confirmed that displaying STAND's instance certainty measure was useful for assessing the AI2T agent's learning progress toward 100% completeness. Several participants in study 2 indicated that this indicator influenced their decision of when to stop training the agent on new problems. This is a strong preliminary indication that the certainty score indicators had the intended effect. A future randomized experiment would be able to lend stronger statistical evidence for the connection between the availability of this indicator and high authoring completeness. However, the productive monotonicity measure we report in our simulation experiments already establishes that this measure accurately reflects agent learning, so it is reasonable to conclude that if users were explicitly trained to interpret it, they could use it successfully as a heuristic for estimating holdout completeness.

In study 2, when users did not achieve 100% model-tracing completeness they either made clear mistakes during authoring (e.g. users 5 and 8) or trained AI2T on too few problems. Thus, improving AI2T's robustness may largely come down to better support for training users and helping them catch mistakes. Participants 3, 4, and 6 likely fell short of 100% because they trained AI2T on too few problems in fraction arithmetic (we did not identify any uncaught mistakes upon reviewing their screen recordings). Participant 4 was the only user among these three who claimed to use certainty scores, and the only participant familiar with the trajectory of the AI2T project—specifically that AI2T had become more data-efficient in the months prior to study 2—and thus they may have had a skewed belief of how little training was required. When participants strictly interpreted a less than $< 100\%$ certainty score as an indication of incompleteness they trained AI2T on a sufficient number of problems. Consequently, a little bit of training to check for mistakes, and correctly interpret certainty scores may go a long way toward helping authors use AI2T effectively.

10.5 Who can use AI2T?

The major aim of this work was to prototype a method whereby the authoring of complex ITSs is made simple, fast, and accessible to non-programmers. Instructional designers, learning engineers, teachers, and researchers are all professionals who may or may not have programming expertise, but who could all benefit from being able to author complex ITSs quickly. Many of our participants' backgrounds align with the more likely populations that would use a tool like AI2T: instructional designers, learning engineers, and researchers. While teachers are a population of interest, a prevailing theme in our conversations with K-12 teachers has been that curating practice materials is beyond the scope of their job description and an unwelcome burden on top of their already busy schedules. Perhaps AI2T is a convenient enough tool to soften that position. Albeit, our broader aim of simplifying the authoring of incisive and complex tutor adaptivity is arguably better suited to professionals with the bandwidth to commit that complexity to experimentation.

11 Future Work

11.1 Design Support for Open-ended Authoring

Our focus of this work was AI2T's usability, but more open-ended authoring evaluations may shed light on the unique needs and design perspectives of in-service professionals. ITS authoring typically requires authors to deliberately design interfaces around adaptive step-by-step support, and this design process typically involves a design loop: a cyclic process of classroom testing and revision [3]. A common mistake in this process is designing instruction around final problem solutions or problem-solving with too large of steps that do not sufficiently break down strategies into their most fine-grained elements. AI2T may play a beneficial role in supporting more adaptive designs earlier in development. Much like students with low-prior knowledge who benefit from bite-sized instruction that reduces cognitive load [31, 47, 57], AI2T tends to learn more effectively from granular step-by-step instruction. Prior work has demonstrated that this feature of simulated learners can be useful for automated student model discovery [38], and as a tool for cognitive task analysis [44]. Computational models of learning like AI2T that model the general processes by which students' knowledge structures change throughout learning have broad benefits toward furthering the science of learning and for analyzing individual learning tasks [25, 41, 65] that go well beyond the affordances of traditional posthoc analyses of student performance data [11].

11.2 Broader Authoring with AI2T

One challenge of using AI2T for general-purpose authoring is that it depends upon a set of primitives in its function library to explain and generalize from users' demonstrated actions. A great deal can be authored with AI2T's current primitive library. In addition to the two domains we focus on in our user studies, we have replicated dozens of Cognitive Tutors and Apprentice Tutors with AI2T. Nonetheless, any fixed library has constraints. As we have previously mentioned, LLMs could play a role in softening that limitation. It seems plausible that an author could "vibe code" missing primitives without leaving AI2T's interface.

Adding yet more performance and learning mechanisms to AI2T could also expand the scope of what it can be used to author with simple primitive functions. For instance, Li et. al. added a representation learning mechanism to SimStudent [39] that greatly simplified the primitive functions it needed to learn algebra equation-solving skills. There are similar opportunities along these lines for using pre-trained generative AI to parse content like text and images into a structured representation that AI2T can reason over. These features could extend AI2T's authoring scope to include domains with word problems and problems that involve reasoning over figures.

11.3 Supports for Finding and Fixing Mistakes

Two of our study 2 participants made training mistakes that produced errors that they did not identify and fix, leading to very low final completeness scores. One of the more impactful varieties of mistakes was teaching AI2T an incorrect skill from an incorrect demonstration, or as a result of bad explanations with incorrect formulae or arguments. Since bad skills are relatively easy to identify, eliminating this kind of mistake could be supported by simply adding a feature for removing or editing whole skills, instead of requiring users to delete all of their supporting examples manually.

Mistakes of mislabelling actions' correctness is a harder category of mistake to track down. However, STAND may provide a path toward a solution. Internally STAND filters examples into bins of common examples (like leaves in a decision tree) [66]. It may well be that mistakes and edge cases tend to filter into bins isolated from the rest. If this is the case, it may be possible for STAND to accurately suggest training examples that it suspects are mistakes.

11.4 Reintroducing Code-Checking

This work has almost entirely avoided displaying AI2T's internal knowledge representations back to users—they do not see the HTNs or skill preconditions that it induces. We have done this in part to illustrate the possibility of a dynamic between humans and teachable AI, whereby trust in the AI's behavior can be established without verifying its synthesized code. However, a powerful affordance of AI that learns symbolic representations is that sufficiently trained users can, in principle, debug them. By forbidding this possibility outright AI2T falls far to one extreme of a spectrum between non-programming and purely programming-based interactions. Yet, the space in between is ripe with possibility. In the future, AI2T may display its internal knowledge back to users, and they may in turn edit that knowledge or describe new knowledge back to AI2T in turn.

12 Conclusion

Well beyond prior attempts at implementing authoring-by-tutoring [43, 45, 63] this work has demonstrated a path towards methods of ITS authoring that are approachable for non-programmers, yet enable the authoring of flexible and robust ITS behaviors that are typically only implementable with hand-programmed rules. Prior works with SimStudent and AL have only shown imperfect ITS induction [63, 64] even in the hands of their creators [43, 45, 64]. By

contrast, our evaluations of AI2T show half of our untrained participants, who were mostly non-programmers, succeeding at producing model-tracing complete ITSs with authoring-by-tutoring, and several others achieving nearly 100% model-tracing completeness. This work presents several interaction design considerations for future authoring-by-tutoring work, including methods for verifying and annotating demonstrations, visualizing and navigating between solution paths, and displaying agents' certainty of proposed actions. STAND [66] and our approach to HTN induction from action sequences mark major machine learning improvements over prior authoring-by-tutoring systems. They enable more data-efficient and robust interactive induction. One of our more surprising results is that STAND's *instance certainty* measure could predict improvements in holdout set performance far better than common ensemble methods like XGBoost, and was used successfully by many of our participants to determine when AI2T had been trained on a sufficient number of problems.

References

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

[2] Vincent Aleven, Bruce M McLaren, Jonathan Sewall, Martin Van Velsen, Octav Popescu, Sandra Demi, Michael Ringenberg, and Kenneth R Koedinger. 2016. Example-Tracing Tutors: Intelligent Tutor Development for Non-Programmers. *International Journal of Artificial Intelligence in Education* 26, 1 (2016), 224–269.

[3] Vincent Aleven, Elizabeth A McLaughlin, R Amos Glenn, and Kenneth R Koedinger. 2016. Instruction based on adaptive learning technologies. *Handbook of research on learning and instruction* 2 (2016), 522–560.

[4] Ankuj Arora, Humbert Fiorino, Damien Pellier, Marc Métivier, and Sylvie Pesty. 2018. A review of learning planning action models. *The Knowledge Engineering Review* 33 (2018), e20.

[5] Stephen B Blessing. 1997. A programming by demonstration authoring tool for model-tracing tutors. *International Journal of Artificial Intelligence in Education* 8 (1997), 233–261.

[6] Leo Breiman. 1996. Bagging predictors. *Machine learning* 24 (1996), 123–140.

[7] Leo Breiman. 2001. Random forests. *Machine learning* 45 (2001), 5–32.

[8] Leo Breiman. 2017. *Classification and Regression Trees*. Routledge.

[9] Tommaso Calo and Christopher Maclellan. 2024. Towards Educator-Driven Tutor Authoring: Generative AI Approaches for Creating Intelligent Tutor Interfaces. In *Proceedings of the Eleventh ACM Conference on Learning@ Scale*. 305–309.

[10] Paulo Carvalho, Napol Rachatasumrit, and Kenneth R Koedinger. 2022. Learning depends on knowledge: The benefits of retrieval practice vary for facts and skills. In *Proceedings of the Annual Meeting of the Cognitive Science Society*, Vol. 44.

[11] Hao Cen, Kenneth Koedinger, and Brian Junker. 2006. Learning Factors Analysis–A General Method for Cognitive Model Evaluation and Improvement. In *International Conference on Intelligent Tutoring Systems*. Springer, 164–175.

[12] Tianqi Chen and Carlos Guestrin. 2016. XGBoost: A scalable tree boosting system. In *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*. 785–794.

[13] Michelene TH Chi and Ruth Wylie. 2014. The ICAP framework: Linking cognitive engagement to active learning outcomes. *Educational psychologist* 49, 4 (2014), 219–243.

[14] Chase C Cicchetti. 2024. AI in higher education: Does not help, might hurt. *Journal of Education for Business* 99, 7-8 (2024), 438–443.

[15] Richard E Clark, David F Feldon, Jeroen JG Van Merrienboer, Kenneth A Yates, and Sean Early. 2008. Cognitive task analysis. In *Handbook of research on educational communications and technology*. Routledge, 577–593.

[16] Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, et al. 2021. Training verifiers to solve math word problems, 2021. *URL https://arxiv. org/abs/2110.14168* (2021).

[17] Albert T Corbett and John R Anderson. 1994. Knowledge tracing: Modeling the acquisition of procedural knowledge. *User modeling and user-adapted interaction* 4, 4 (1994), 253–278.

[18] Allen Cypher and Daniel Conrad Halbert. 1993. *Watch What I Do: Programming by Demonstration*. MIT press.

[19] Jerry Alan Fails and Dan R Olsen Jr. 2003. Interactive machine learning. In *Proceedings of the 8th international conference on Intelligent user interfaces*. 39–45.

[20] Arthur C Graesser, Mark W Conley, and Andrew Olney. 2012. Intelligent tutoring systems. (2012).

[21] Sumit Gulwani, William R Harris, and Rishabh Singh. 2012. Spreadsheet data manipulation using examples. *Commun. ACM* 55, 8 (2012), 97–105.

[22] Yu Guo, Joseph E Beck, and Neil T Heffernan. 2008. Trying to Reduce Bottom-out hinting: Will telling student how many hints they have left help?. In *International Conference on Intelligent Tutoring Systems*. Springer, 774–778.

[23] Adit Gupta, Jennifer Reddig, Tommaso Calo, Daniel Weitekamp, and Christopher J MacLellan. 2025. Beyond Final Answers: Evaluating Large Language Models for Math Tutoring. *arXiv preprint arXiv:2503.16460* (2025).

[24] Adit Gupta, Momin Siddiqui, Glen Smith, Jenn Reddig, and Christopher MacLellan. 2024. Intelligent Tutors for Adult Learners: An Analysis of Needs and Challenges. *arXiv preprint arXiv:2412.04477* (2024).

[25] Erik Harpstead, Christopher J MacLellan, Daniel Weitekamp, and Kenneth R Koedinger. [n. d.]. The use simulated learners in adaptive education. *AIAED-19: AI+ Adaptive Education* ([n. d.]), 1–3.

[26] Neil T Heffernan and Cristina Lindquist Heffernan. 2014. The ASSISTments ecosystem: Building a platform that brings scientists and teachers together for minimally invasive research on human learning and teaching. *International Journal of Artificial Intelligence in Education* 24, 4 (2014), 470–497.

[27] Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. 2021. Measuring mathematical problem solving with the math dataset. *arXiv preprint arXiv:2103.03874* (2021).

[28] Chad Hogg, Héctor Munoz-Avila, and Ugur Kuter. 2008. HTN-MAKER: Learning HTNs with Minimal Additional Knowledge Engineering Required.. In *AAAI*. 950–956.

[29] Scott B Huffman and John E Laird. 1995. Flexibly instructable agents. *Journal of Artificial Intelligence Research* 3 (1995), 271–324.

[30] Viswanathan Kodaganallur, Rob R Weitz, and David Rosenthal. 2005. A comparison of model-tracing and constraint-based intelligent tutoring paradigms. *International Journal of Artificial Intelligence in Education* 15, 2 (2005), 117–144.

[31] Kenneth R Koedinger and Vincent Aleven. 2007. Exploring the assistance dilemma in experiments with cognitive tutors. *Educational Psychology Review* 19 (2007), 239–264.

[32] Kenneth R Koedinger, Jihee Kim, Julianna Zhuxin Jia, Elizabeth A McLaughlin, and Norman L Bier. 2015. Learning is not a spectator sport: Doing is better than watching for learning from a MOOC. In *Proceedings of the second (2015) ACM conference on learning@ scale*. 111–120.

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

[34] John E Laird, Kevin Gluck, John Anderson, Kenneth D Forbus, Odest Chadwicke Jenkins, Christian Lebiere, Dario Salvucci, Matthias Scheutz, Andrea Thomaz, Greg Trafton, Robert E Wray, Shiwali Mohan, and James R Kirk. 2017. Interactive Task Learning. *IEEE Intelligent Systems* 32, 4 (2017), 6–21. https://doi.org/10.1109/MIS.2017.3121552

[35] Tessa Lau, Steven A Wolfman, Pedro Domingos, and Daniel S Weld. 2003. Programming by demonstration using version space algebra. *Machine Learning* 53, 1 (2003), 111–156.

[36] Lane Lawley and Christopher Maclellan. 2024. VAL: Interactive Task Learning with GPT Dialog Parsing. In *Proceedings of the CHI Conference on Human Factors in Computing Systems*. 1–18.

[37] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. 2015. Deep learning. *nature* 521, 7553 (2015), 436–444.

[38] Nan Li, William W Cohen, Kenneth R Koedinger, and Noboru Matsuda. 2011. A machine learning approach for automatic student model discovery.. In *Edm*. ERIC, 31–40.

[39] Nan Li, Noboru Matsuda, William W Cohen, and Kenneth R Koedinger. 2015. Integrating representation learning and skill learning in a human-like intelligent agent. *Artificial Intelligence* 219 (2015), 67–91.

[40] Toby Jia-Jun Li, Amos Azaria, and Brad A Myers. 2017. SUGILITE: Creating Multimodal Smartphone Automation by Demonstration. In *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems*. ACM, 6038–6049.

[41] Christopher J MacLellan. 2017. *Computational Models of Human Learning: Applications for Tutor Development, Behavior Prediction, and Theory Testing*. Ph. D. Dissertation. Carnegie Mellon University.

[42] Christopher J Maclellan, Erik Harpstead, Rony Patel, and Kenneth R Koedinger. 2016. The Apprentice Learner Architecture: Closing the Loop between Learning Theory and Educational Data. *International Educational Data Mining Society* (2016).

[43] Christopher J MacLellan and Kenneth R Koedinger. 2020. Domain-General Tutor Authoring with Apprentice Learner Models. *International Journal of Artificial Intelligence in Education* (2020), 1–42.

[44] Noboru Matsuda. 2022. Teachable agent as an interactive tool for cognitive task analysis: A case study for authoring an expert model. *International Journal of Artificial Intelligence in Education* 32, 1 (2022), 48–75.

[45] Noboru Matsuda, William W Cohen, and Kenneth R Koedinger. 2015. Teaching the Teacher: Tutoring SimStudent Leads to More Effective Cognitive Tutor Authoring. *International Journal of Artificial Intelligence in Education* 25, 1 (2015),

1–34.
[46] Richard G McDaniel and Brad A Myers. 1997. Gamut: demonstrating whole applications. In *Proceedings of the 10th annual ACM symposium on User interface software and technology*. 81–82.
[47] Katherine L McNeill, David J Lizotte, Joseph Krajcik, and Ronald W Marx. 2006. Supporting students' construction of scientific explanations by fading scaffolds in instructional materials. *The Journal of the Learning Sciences* 15, 2 (2006), 153–191.
[48] Iman Mirzadeh, Keivan Alizadeh, Hooman Shahrokhi, Oncel Tuzel, Samy Bengio, and Mehrdad Farajtabar. 2024. Gsm-symbolic: Understanding the limitations of mathematical reasoning in large language models. *arXiv preprint arXiv:2410.05229* (2024).
[49] Tom M Mitchell. 1982. Generalization as Search. *Artificial Intelligence* 18, 2 (1982), 203–226.
[50] Stephen H Muggleton, Dianhuan Lin, and Alireza Tamaddoni-Nezhad. 2015. Meta-interpretive learning of higher-order dyadic datalog: Predicate invention revisited. *Machine Learning* 100, 1 (2015), 49–73.
[51] Brad A Myers. 1988. *Creating user interfaces by demonstration*. Academic Press Professional, Inc.
[52] Hyacinth S Nwana. 1990. Intelligent tutoring systems: an overview. *Artificial Intelligence Review* 4, 4 (1990), 251–277.
[53] Benjamin D Nye, Dillon Mee, and Mark G Core. 2023. Generative Large Language Models for Dialog-Based Tutoring: An Early Consideration of Opportunities and Concerns.. In *LLM@ AIED*. 78–88.
[54] Scott Ososky, Keith Brawner, Benjamin Goldberg, and Robert Sottilare. 2016. GIFT Cloud: Improving usability of adaptive tutor authoring tools within a web-based application. In *Proceedings of the Human Factors and Ergonomics Society Annual Meeting*, Vol. 60. SAGE Publications Sage CA: Los Angeles, CA, 1389–1393.
[55] Zachary A Pardos, Matthew Tang, Ioannis Anastasopoulos, Shreya K Sheel, and Ethan Zhang. 2023. Oatutor: An open-source adaptive tutoring system and curated content library for learning sciences research. In *Proceedings of the 2023 chi conference on human factors in computing systems*. 1–17.
[56] Rony Patel, Ran Liu, and Kenneth R Koedinger. 2016. When to Block versus Interleave Practice? Evidence Against Teaching Fraction Addition before Fraction Multiplication.. In *CogSci*.
[57] Sadhana Puntambekar and Roland Hubscher. 2005. Tools for scaffolding students in a complex learning environment: What have we gained and what have we missed? *Educational psychologist* 40, 1 (2005), 1–12.
[58] Joshua S Rule, Steven T Piantadosi, Andrew Cropper, Kevin Ellis, Maxwell Nye, and Joshua B Tenenbaum. 2024. Symbolic metaprogram search improves learning efficiency and explains rule learning in humans. *Nature Communications* 15, 1 (2024), 6847.
[59] Momin N Siddiqui, Adit Gupta, Jennifer M Reddig, and Christopher J MacLellan. 2024. HTN-Based Tutors: A New Intelligent Tutoring Framework Based on Hierarchical Task Networks. In *Proceedings of the Eleventh ACM Conference on Learning@ Scale*. 491–495.
[60] Kurt VanLehn. 1987. Learning one subprocedure per lesson. *Artificial Intelligence* 31, 1 (1987), 1–40.
[61] Kurt VanLehn. 1990. *Mind bugs: The origins of procedural misconceptions*. MIT press.
[62] Kurt VanLehn. 2011. The relative effectiveness of human tutoring, intelligent tutoring systems, and other tutoring systems. *Educational Psychologist* 46, 4 (2011), 197–221.
[63] Daniel Weitekamp, Erik Harpstead, and Kenneth Koedinger. 2020 in press. An Interaction Design for Machine Teaching to Develop AI Tutors. *CHI* (2020 in press).
[64] Daniel Weitekamp, Erik Harpstead, and Kenneth Koedinger. 2021. Toward Stable Asymptotic Learning with Simulated Learners. In *International Conference on Artificial Intelligence in Education*. Springer, 390–394.
[65] Daniel Weitekamp and Kenneth Koedinger. 2023. Computational models of learning: Deepening care and carefulness in AI in education. In *International Conference on Artificial Intelligence in Education*. Springer, 13–25.
[66] Daniel Weitekamp and Kenneth Koedinger. 2024. STAND: Data-Efficient and Self-Aware Precondition Induction for Interactive Task Learning. arXiv:2409.07653 [cs.LG] https://arxiv.org/abs/2409.07653
[67] Daniel Weitekamp III, Erik Harpstead, Christopher J MacLellan, Napol Rachatasumrit, and Kenneth R Koedinger. 2019. Toward Near Zero-Parameter Prediction Using a Computational Model of Student Learning. *International Educational Data Mining Society* (2019).
[68] Kenneth Yates, Maura Sullivan, and Richard Clark. 2012. Integrated studies on the use of cognitive task analysis to capture surgical expertise for central venous catheter placement and open cricothyrotomy. *The American journal of surgery* 203, 1 (2012), 76–80.
[69] Xiaojin Zhu. 2015. Machine teaching: An inverse problem to machine learning and an approach toward optimal education. In *Proceedings of the AAAI conference on artificial intelligence*, Vol. 29.